A dissertation submitted in partial fulfilment of the requirements for
M.Tech(Computer Science) degree of the Indian Statistical Institute

# Integer Linear Programming Based Scheduling for H.264 Video Decoding in Multicore Processor

**M.Tech(Computer Science) Dissertation Report**

**By**


## Somabrata Pramanik

**Roll Number : CS0914**


**Under the supervision of**


## Dr. Susmita Sur-Kolay

**and**

## Dr. Ansuman Banerjee

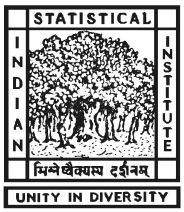
**Advanced Computing and Microelectronics Unit**




Indian Statistical Institute

203, B.T. Road

Kolkata 700 108




# Indian Statistical Institute

Kolkata 700 108

## CERTIFICATE

This is to certify that the thesis entitled "Integer Linear Programming Based Scheduling for H.264 Video Decoding in Multicore processor" is submitted in the partial fulfilment of the degree of M. Tech. in Computer Science at Indian Statistical Institute, Kolkata. It is fully adequate in scope and quality as a dissertation for the required degree.

The thesis is a faithfully record of bonafide research work carried out by Somabrata Pramanik under our supervision and guidance.

Dr. Susmita Sur-Kolay
(Supervisor)
ACMU

Dr. Ansuman Banerjee
(Co-supervisor)
ACMU

Countersigned
(External Examiner)
Date:

# Acknowledgement

*I would like to express my sincere gratitude to the supervisor of this study, Dr. Susmita Sur-Kolay and Dr. Ansuman Banerjee. I also take this opportunity to express my gratitude to Mr. Bhaskar Karmakar and Mr. Prasenjit Basu of Texas Instruments, Bangalore. Their command over this matter has been a great help for my analysis. They helped me in all aspects including the preparation of this manuscript. This work has been possible only because of their continuous suggestions, inspiration, motivation and full freedom given to me to incorporate my ideas. I also take this opportunity to thank all my teachers who have taught me in my M. Tech. course and last but not the least I thank all my family members and friends for their endless support.*

Place : Kolkata
Date : 19.07.2011

Somabrata Pramanik

# Contents

# Chapter 1

# Introduction

Demand of high quality video based technologies are increasing with high-definition televisions, video streaming through internet and many other applications. Compression ratio of the previous standards are not enough for these upcoming technologies. The latest video compression standard, ITU-T recommended H.264/AVC (also known as ISO/IEC 14496 (MPEG-4) Part 10 for Advanced Video Coding) is expected to become the video standard of choice in the coming years for its higher compression ratio and use of more efficient technologies.

H.264/AVC is an open, licensed standard that supports the most efficient video compression techniques available today. The average bit rate reduction by H.264 encoder is of 80% compared to the Motion JPEG format and 50% the MPEG-4 Part 2 standard, without compromising the image quality. This means, much less network bandwidth and storage space are required for a video file; or in another way, much higher video quality can be achieved for a given bit rate.

## 1.1 Motivation

H.264/AVC is very appropriate for the applications like multimedia streaming, high quality video broadcasting, video storage in optical and magnetic discs. But, these applications requires high speed encoding and decoding of video data. H.264/AVC encoder and decoder both have a sequential, data dependent flow of execution. This property makes it difficult to leverage the potential performance gain that could be achieved by the use of emerging many core processors.

Dedicated silicon implementations of H.264/AVC codecs are presently available which can perform 30fps encode/decode for 1080p video sequences. Hardware implementations for each new video compression standard on different platform is costly enough. That is why we need a parallel software implementation of H.264/AVC codec that can perform as efficiently as the hardware implemen-

tation and can run on different hardware platforms. If the hardware platform changes the software implementation needs smaller amount of change than dedicated silicon implementation.

## 1.2 Scope

In this project, we consider an H.264 decoder and explore the possibilities of parallelism. There are a couple of reasons behind taking up the decoder (and not the encoder) as part of this parallelization effort. Firstly, the encoding problem is a natively parallel one, and hence, lends itself more naturally to a parallelized execution environment and there are already numerous successful attempts in this direction. However, the decoding algorithm poses certain challenges to parallelization. Secondly, there is a decoding step inside the encoder as well and therefore, any success in parallelizing the decoder would naturally expedite the encoder as well.

The key task in parallelization of the H.264 decoder is to find a scheduler, which can distribute the decoder flow of execution into several cores efficiently. The scheduler must consider data dependency issues as well as inter-communication and synchronization between the processing units. There are many proposed schedulers for this core allocation, using different strategies. Performance of these schedulers depends on the scalability and hardware utilization it can achieve.

Our objective is to find a scheduler which can allocate the processor cores for decoding in most efficient way so that the time to decode is optimized. As this scheduler is an optimized one, this can also be used to measure the performance of other non optimized schedulers.

## 1.3 Organisation

This report is organised as follows. In Chapter 2 related works on parallelizing h.264 is discussed briefly. In Chapter 3 our primary objectives and problem formulation is discussed along with a high level overview of H.264 decoder and the data structure behind it. Chapter 4 provides the study on various modes of intra macroblock prediction and corresponding data dependencies. Chapter 5 showing how a H.264 decoder works on various cache environments. Three formulation of Integer Linear Programming to minimize the decoding time in a scheduler is given in chapter 6.

# Chapter 2

# Review of Earlier Works

To incorporate with chip multi-processors, the H.264 codec needs thread level parallelism. Sufficient scalability can be achieved by macroblock level parallelization, so that inter-frame MBs and intra frame MBs can be processed simultaneously if they have their sufficient information to be decoded.
The following are abstract of some papers on parallelizing the decoding process for multi-core processors.

## 2.1 Parallel Scalability of H.264

**Authors:** Cor Meenderinck, Arnaldo Azevedo, Mauricio Alvarez, Ben Juurlink, and Alex Ramirez.

**Affiliation:** Delft University, University of Catalonia, Barcelona Supercomputing Center.

**Abstract:** An important question is whether emerging and future applications exhibit sufficient parallelism, in particular thread-level parallelism (TLP), to exploit the large numbers of cores future CMPs are expected to contain. As a case study we investigate the parallel scalability of the H.264 decoding process. Previously proposed parallelization strategies such as slice-level, frame-level, and intra-frame macroblock (MB) level parallelism, are not sufficiently scalable. We therefore propose a novel strategy, called 3D-Wave, which is mainly based on the observation that inter-frame dependencies have a limited spatial range. Because of this, certain MBs of consecutive frames can be decoded in parallel. The 3D-Wave strategy allows 4000 to 9000 MBs to be processed in parallel, depending on the input sequence. We also perform a case study to assess the practical value and possibilities of the 3D-Wave strategy. The results show that our strategy provides sufficient parallelism to efficiently exploit the capabilities of future manycore CMPs.

**Summary:** Describes and compares the different Data level decompositions possible on H.264 decoder. Frame level parallelism has very little scalability since there are very few B frames between P frames. In H,264 B frames can also be used as reference. This reduces FLP even more. Slice level parallelism depends on the size and number of slices/frame. These parameters are dependent on the encoder. Therefore load balancing and low scalability are its issues. In MB level parallelism two strategies, 2-D and 3-D wave are discussed. This work tries to combine FLP with MLP dynamically and measure performance improvements. Study reveals 3-D wave provides huge number of parallel MBs and requires large memory bandwidth which is even beyond the scope of future many core CMPs. So the performance of 3-D is studied under limited resource availability conditions.

**Advantages:** Even with limited resources 3-D wave has the potential of utilizing the computational power available provided sufficient memory bandwidth to support constant number of frames in flight.

## 2.2 A Highly Scalable Parallel Implementation of H.264

**Authors:** Arnaldo Azevedo, Ben Juurlink, Cor Meenderinck, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, Alex Ramirez, Mateo Valero.

**Affiliation:** Delft University, University of Catalonia, Barcelona Supercomputing Center.

**Abstract:** Developing parallel applications that can harness and efficiently use future many-core architectures is the key challenge for scalable computing systems. We contribute to this challenge by presenting a parallel implementation of H.264 that scales to a large number of cores. The algorithm exploits the fact that independent macroblocks (MBs) can be processed in parallel, but whereas a previous approach exploits only intra-frame MB-level parallelism, our algorithm exploits intra-frame as well as inter- frame MB-level parallelism. It is based on the observation that inter-frame dependencies have a limited spatial range. The algorithm has been implemented on a many-core architecture consisting of NXP TriMedia TM3270 embedded processors. This required to develop a subscription mechanism, where MBs are subscribed to the kick-off lists associated with the reference MBs. Extensive simulation results show that the implementation scales very well, achieving a speedup of more than 54 on a 64-core processor, in which case the previous approach achieves a speedup of only 23. Potential drawbacks of the 3D-Wave strategy are that the memory requirements increase since there can be many frames in flight, and that the frame latency might increase. Scheduling policies to address these drawbacks are also presented. The results show that these policies combat memory and latency

issues with a negligible effect on the performance scalability. Results analysing the impact of the memory latency, L1 cache size, and the synchronization and thread management overhead are also presented. Finally, we present performance requirements for entropy (CABAC) decoding.

**Summary:** This paper describes the concept of 3-D wave, how it is implemented, how its performance gain is much more than 2-D wave technique. The performance of 2-D wave is highly affected by the frame resolution and scalability is not constant throughout the decoding of a frame. To overcome these drawbacks 3-D technique is exploited along with 2-D technique. It supports hundreds of frames in flight and thousands of MBs to be processed in parallel given sufficient memory bandwidth and a good frame scheduling policy.

**Advantages:**

- Scalability is much better than 2-D.
- With a good Frame scheduling and priority mechanism it is possible to achieve almost the performance achieved by original 3-D and minimize Frame latency. Original 3-D needs unlimited memory but FS and priority scheduling makes it possible to achieve it with limited memory.

**Disadvantages:**

- Presently the subscribe MBs are chosen statically so the number of frames in flight is almost constant. It should be chosen dynamically.
- The scalability of 3-D decreases for large number of cores because cache thrashing occurs The scalability also decreases at the start or end of a sequence since little parallelism is available at that time.
- For higher resolution video, no relevant speedup is achieved unlike 2-D.
- FS and FP achieve lower frame latency and higher scalability at the cost of increased memory traffic.
- Increase in average memory latency decreases performance gain (AML of 40-50 cycles is ok). L1 cache should be at least 64KB for higher number of cores.
- The overhead of TLP must be limited to maximum 30% of the MB decoding time. Otherwise the performance degenerates drastically with increased number of cores.

## 2.3 Parallel H.264 Decoding on an Embedded Multicore Processor

**Authors:** Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, Andrei Terechko, Jan Hoogerbrugge, Mauricio Alvarez, Alex Ramirez, Mateo Valero.

**Affiliation:** Delft University, University of Catalonia, Barcelona Supercomputing Center.

**Abstract:** In previous work the 3D-Wave parallelization strategy was proposed to increase the parallel scalability of H.264 video decoding. This strategy is based on the observation that inter-frame dependencies have a limited spatial range. The previous results, however, investigate application scalability on an idealized multiprocessor. This work presents an implementation of the 3D-Wave strategy on a multicore architecture composed of NXP TriMedia TM3270 embedded processors. Results show that the parallel H.264 implementation scales very well, achieving a speedup of more than 54 on a 64-core processor. Potential drawbacks of the 3D-Wave strategy are that the memory requirements increase since there can be many frames in flight, and that the latencies of some frames might increase. To address these drawbacks, policies to reduce the number of frames in flight and the frame latency are also presented. The results show that our policies combat memory and latency issues with a negligible effect on the performance scalability.

**Summary:** Describes the 3-D and 2-D implementation process. Shows 3-D is much more scalable than 2- D. Scalability of 2-D largely depends on the resolution of video. 3-D is free from this drawback. Once the reference MB of a MB in another frame is completed the current MB may start processing. But this may lead to large number of Frames on the fly and demand large memory bandwidth. This problem is solved by introducing the concept of subscription MB which limits the number of Frames in flight. A Frame scheduling policy which introduces frame priority, helps reducing the frame latency. 3-D is implemented on an existing 2-D implementation. A feature called Tail submit is used to minimize the spatial dependency of Mbs and reduce memory read/writes.

**Advantages and Disadvantages:** same as section 2.2

## 2.4 Efficient Parallelization Of H.264 Decoding With Macro Block Level Scheduling

**Authors:** Jike Chong, Nadathur Satish, Bryan Catanzaro, Kaushik Ravindran, Kurt Keutzer.

**Affiliation:** University of California, Berkeley.

**Abstract:** The H.264 decoder has a sequential, control intensive front end that makes it difficult to leverage the potential performance of emerging many core processors. Preparsing is a functional parallelization technique to resolve this front end bottleneck. However, the resulting parallel macro block (MB) rendering tasks have highly input-dependent execution times and precedence

constraints, which make them difficult to schedule efficiently on many core processors. To address these issues, we propose a two step approach:

(i) a custom pre-parsing technique to resolve control dependencies in the input stream and expose MB level data parallelism.

(ii) an MB level scheduling technique to allocate and load balance MB rendering tasks.

The run time MB level scheduling increases the efficiency of parallel execution in the rest of the H.264 decoder. It provides 60% speedup over greedy dynamic scheduling and 9-15% speedup over static compile time scheduling for more than four processors. The pre-parsing technique coupled with run time MB level scheduling enables a potential 7x speedup for H.264 decoding.

**Summary:** Each video frame goes through 3 stages- parse, render and filter. This paper concentrates on parallelizing the first 2 stages. Parsing is a major bottleneck in parallelization. To overcome this, a pre- parsing strategy has been applied and some of recently pre-parsed frames are buffered and later stages are performed on them in parallel. This paper compares different MB scheduling strategies greedy dynamic, static compile time and run time. Among the three greedy dynamic fails to preserve MB dependencies, static compile time fails to handle input dependent data. The run time MB scheduling preserves both.

**Advantages:** The runtime scheduling matches the ideal scheduling most closely

## 2.5 A Highly Efficient Parallel Algorithm for H.264 Encoder Based on Macroblock Region Partition

**Authors:** Shuwei Sun, Dong Wang, and Shuming Chen.

**Affiliation:** National Institute of Defense Technology, China.

**Abstract:** This paper proposes a highly efficient MBRP parallel algorithm for H.264 encoder, which is based on the analysis of data dependencies in H.264 encoder. In the algorithm, the video frames are partitioned into several MB regions, each of which consists of several adjoining columns of macro-blocks (MB), which could be encoded by one processor of a multi-processor system. While starting up the encoding process, the wave-front technique is adopted, and the processors begin encoding process orderly. In the MBRP parallel algorithm, the quantity of data that needs to be exchanged between processors is small, and the loads in different processors are balanced. The algorithm could efficiently encode the video sequence without any influence on the compression ratio. Simulation results show that the proposed MBRP parallel algorithm can achieve

higher speedups compared to previous approaches, and the encoding quality is the same as JM 10.2.

**Summary:** This paper identifies different data dependencies between Mbs:(a)for inter prediction dependencies between Mbs belonging to different frames and (b)between Mbs in the same frame for Intra prediction. This paper introduces a MBRP algorithm. It partitions a frame into partitions consisting of consecutive columns. Each partition is then assigned to a separate processor. The algorithm proceeds by wave front method.

**Advantages:**

- Each partition contains approximately equal number of consecutive non-overlapping columns. So load balancing is achieved easily.
- Amount of data exchanged between processors is smaller.

**Disadvantages:**

- Each partition must contain at least 2 consecutive MB columns. So highest speedup possible is restricted by the resolution of the video source.

## 2.6 Novel Approaches to Parallel H.264 Decoder on Symmetric Multicore Systems

**Authors:** Kue-Hwan Sihn, Hyunki Baik, Jong-Tae Kim, Sehyun Bae, Hyo Jung Song.

**Affiliation:** Software Lab., SAIT, Samsung Electronics,Visual Display Division, Samsung Electronics.

**Abstract:** Novel approaches to parallel H.264 decoder for symmetric multi-core processors are presented. The basic partitioning of the decoder is coarse-grained and hybrid method of the data partitioning and functional partitioning. We investigate the performance bottleneck of the parallelized decoder, and propose two new approaches, software memory throttling and fair load balancing. The software memory throttling limits the number of cores involved in the parallel motion compensation to achieve better speedup and power-saving. The fair load balancing for de-blocking filter reduces load imbalance caused by the conventional static partitioning method. From the evaluation on two different symmetric multi-core platforms, proposed approaches show up to 24% of speedup when there is much bandwidth contention.

**Summary:** A pipeline structure is developed where VLD is overlapped with MC stage. VLD and MC share the same task queue. These stages behave like the producer consumer model. But the speed of VLD is a bottleneck. The 2-D wavefront technique is used both for Inter and Intra frames. Limited shared memory and off-chip memory bandwidth in multi-core processors are a bottleneck. As the number of cores increase the speedup gradually decrease because the multiple threads that are dependent on VLD occupy much of the shred memory. So speedup of VLD is essential. IP and MC deal with small data partitions and they have small load imbalance. But De-blocking filters process large partitions and so they suffer from heavy load imbalance.

**Advantages:**

- Software memory throttling method tries to meet the gap between VLD and MC processing speeds.
- De-blocking of Inter takes 3 times times as compared to that of Intra. While partitioning workload for De-blocking filter we have to consider the Mb-type as well to ensure fair load balancing.

## 2.7 GOP-Level Parallelization of the H.264 Decoder without a Start-Code Scanner

**Authors:** Ahmet GÃijrhanl, Charlie Chung-Ping Chen, Shih-Hao Hung.

**Affiliation:** National Taiwan University.

**Abstract:** Recent researches on parallelization of H.264 video decoders focused on fine-grain methods. These works led to designs having very short latencies and good memory usage. However, they could not reach the scalability of Group of Pictures (GOP) level approaches although assuming a well designed entropy decoder which can feed the increasing number of parallel working cores. We would like to introduce a GOP-level approach due to its high scalability, mentioning solution approaches for the well- known latency and memory issues. Our design revokes the need to a scanner for GOP startcodes which was used in the earlier methods. This approach lets all the cores work on the decoding task. Although the performance on shared memory systems is subject to improve,we have observed a one-to-one linear speedup in parallel working nodes. We have tested our method using a cluster of 5 machines each having 2 processors with 4 cores. The decoding is 5 times faster when we run only one process in each machine, that is we saw one-to-one linear speedup when there is no memory shortage. We observed a maximum of 11 times speedup when using all of the 40 cores distributed among 5 machines.

**Summary:** Each processor is assigned a GOP. GOPs are independent of each other and can be decoded simultaneously. The start point of each GOP is written in the video stream header by the encoder. Each processor in the decoder reads the header and chooses the part it is supposed to decode. This method gives linear speedup in case of small GOP sizes but deteriorates as GOP size increases. For large GOP the memory requirements of the processors increases so more cache pollution occurs. So linear speedup not possible.

**Advantages:**

- Does not need GOP start code scanner

**Disadvantages:**

- Need more memory resource and has long latency problems
- One processor manages the scheduling and the others do parallel decoding. So with P processors only P-1 times maximum speedup possible
- The scheduler may easily become insufficient for feeding the increasing number of parallel working cores.

## 2.8 Scalability of Macroblock-level Parallelism for H.264 Decoding

**Authors:** Mauricio Alvarez Mesa, Alex Ram ÌĄÄśrezy, Arnaldo Azevedoz, Cor Meenderinckz, Ben Juurlinkz, and Mateo Valeroy.

**Affiliation:** Universitat Polit'ecnica de Catalunya, Barcelona, Spain.

**Abstract:** This paper investigates the scalability of MacroBlock (MB) level parallelization of the H.264 decoder for High Definition (HD) applications. The study includes three parts. First, a formal model for predicting the maximum performance that can be obtained taking into account variable processing time of tasks and thread synchronization overhead. Second, an implementation on a real multiprocessor architecture including a comparison of different scheduling strategies and a profiling analysis for identifying the performance bottlenecks. Finally, a trace-driven simulation methodology has been used for identifying the opportunities of acceleration for removing the main bottlenecks. It includes the acceleration potential for the entropy decoding stage and thread synchronization and scheduling. Our study presents a quantitative analysis of the main bottlenecks of the application and estimates the acceleration levels that are required to make the MB-level parallel decoder scalable.

**Summary:** The maximum speedup achievable using MB level parallelism is quite high. But it assumes constant MB processing time and no synchronization overhead. When these factors are taken into account maximum speedup possible is reduced significantly. The paper then compares the various scheduling approaches of Mbs to cores. The static scheduling is good only when MB processing time is constant(which is impossible in real cases). The dynamic scheduling exploits more paralleism but encounters huge overhead due to a centralised task queue read/write. Dynamic scheduling with tail submit is by far the best scheduling approach. Entropy decoding is entirely serial ,so it is decoupled from the parallel part of the decoder and a buffer is inserted between the modules to meet the spped differences of the two parts.

**Advantages:**

- CABAC accelerator improves performance
- Thread synchronization accelerator improves performance

# Chapter 3

# Problem Formulation

## 3.1 H.264 decoder

The H.264 decoder flow of execution on encoded data is as shown in the figure 3.1. Decoder receives the encoded data as NAL unit from Network Abstraction Layer. This encoded data is first entropy decoded to produce transformed and quantised coefficients. These coefficients are then scaled and inverse transformed to regenerate the residual data. Using the header informations already decoded from the bitstream, the decoder creates a prediction block, identical to the original prediction formed in the encoder. This predicted data is added to the residual data, which is already decoded. Then it is filtered to produce each decoded block of a frame or field. In Intra Prediction, the unfiltered data blocks of same frame or field is used to create the prediction block and in Inter Prediction, previously decoded frame or fields are used to predict the current block.

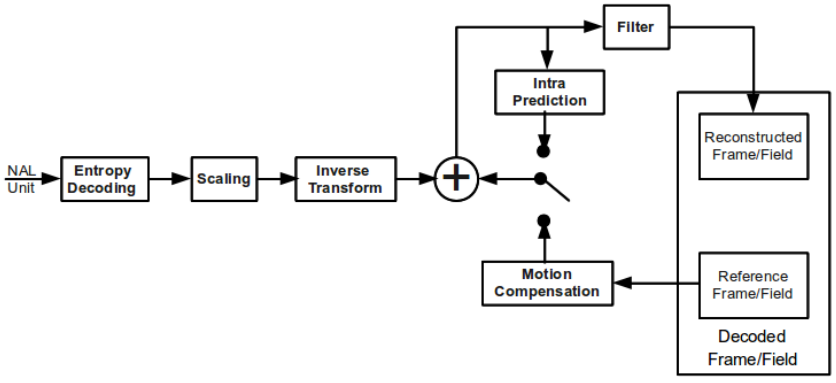



Figure 3.1: *Block Diagram of H.264 Decoder*

## 3.2 H.264/AVC Data Structures

To understand the areas where the H.264 decoder can be parallelized and the limitations in it, we need to know the data structure of H.264/AVC encoded data. A high level overview of the H.264/AVC data structure is given in figure 3.2. The horizontal arrows in this figure specifying a ‘ is a ’ relationship and other arrows specifying a ‘ have a ’ relationship between structures.

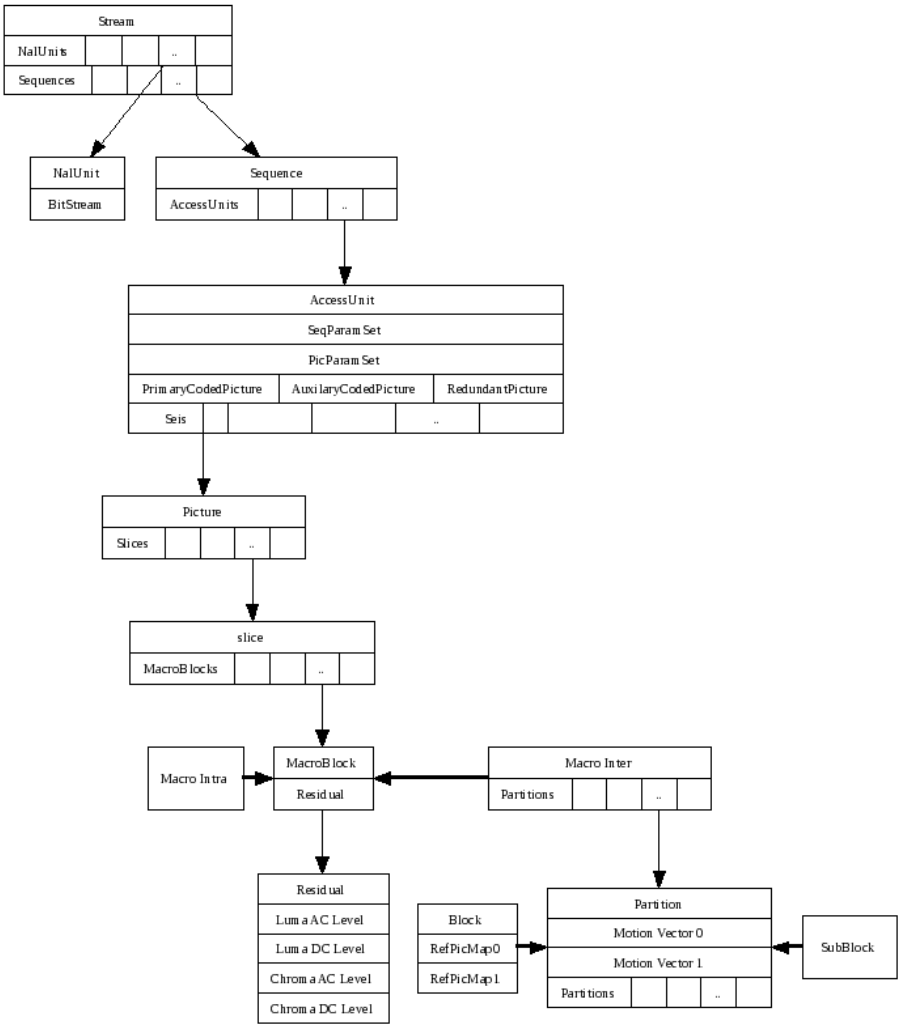



Figure 3.2: *H.264/AVC coded data structure*

**Video Stream:** An encoded H.264 video stream means a collection of organised logical data packets. Each syntax structure in H.264/AVC is placed into one of such packets.

**NAL Unit:** It is raw data packet containing integer number of bytes. The first byte of each NAL unit is a header byte that contains an indication of the type of data in the NAL unit, and the remaining bytes contain payload data of the type indicated by the header. A NAL unit can be of two types,

VCL NAL Unit: contains pixel data of pictures as payload data.

Non VCL NAL Unit: contains other informations which is used to decode an encoded data.

**Video Sequence:** A video stream is generated by a number of video sequences. Each such coded video sequence is a collection of coded pictures called access unit.

**picture Parameter Set:** This is a set of parameters applicable to one or more coded picture of a video sequence. Each VCL NAL unit contains an identifier that refers to the content of the relevant picture parameter set.

**Sequence Parameter Set:** This is a set of parameters applicable to all the coded pictures of a video sequence. In each picture parameter set, there is an identifier that refers to the content of the relevant sequence parameter set.

**Access Unit:** A coded picture is distributed into several NAL Units. An access unit is a collection of such VCL NAL units which is used to decode a single picture. All the VCL NAL units in an access unitare decoded using same sequence parameter set and picture parameter set. This picture is named as primary coded picture. Also there can be an auxiliary coded picture and a redundant picture to recover data loss in the primary coded picture.

**Picture:** A coded picture in a video sequence can be a frame or a single field. This is consisting of a number of slices. Generally, a frame of video can be considered to contain two interleaved fields, a top and a bottom field. The top field contains even-numbered rows and the bottom field contains the odd-numbered rows. If the two fields of a frame were captured at different time instants, the frame is referred to as an interlaced frame, and otherwise it is referred to as a progressive frame.

**Slice:** A picture is divided into one or more parts in H.264/AVC, each such part is called slice. The size of a slice in H.264/AVC is flexible. Each slice of a picture can be decoded separately as there is no data dependency between any two slices of a picture except in the filtering process. Each slice contains a

number of macroblocks.
Depending on the type of macroblocks present in a slice, slices are classified as,

1. **I Slice:** all the macroblocks in this slice are coded using intra prediction.
2. **P Slice:** macroblocks in a P Slice is coded either using intra prediction or using inter prediction with only one motion vector per partition.
3. **B Slice:** all the macroblocks in a B slice is encoded using one of the intra prediction, inter prediction and bi prediction. Bi prediction is a kind of inter prediction, where each partition of a macroblock is predicted using two motion vector to two different reference picture.
4. **SP slice:** A so-called switching P slice that is coded such that efficient switching between different pre-coded pictures becomes possible.
5. **SI slice:** A so-called switching I slice that allows an exact match of a macroblock in an SP slice for random access and error recovery purposes.

**Macroblock:** In H.264/AVC, this is the basic building block of a picture. It can be of two types,

Intra Macroblock: these macroblocks are predicted using intra prediction. This contains residual information after predicction.

Inter macroblock: these macroblocks are predicted using inter prediction. Macroblocks contains the residual information and the information about partitions.

**Partitions:** It can be of two types block and subblock. In both the cases there can be two motion vector specified for motion compensation. In blocks the identification of the reference pictures from where it has to be predicted is mentioned. partitions can be further divided into smaller partitions. Possible Macroblock Partitions and Sub Macroblock Partitions are shown in figure 3.3 and figure 3.4.

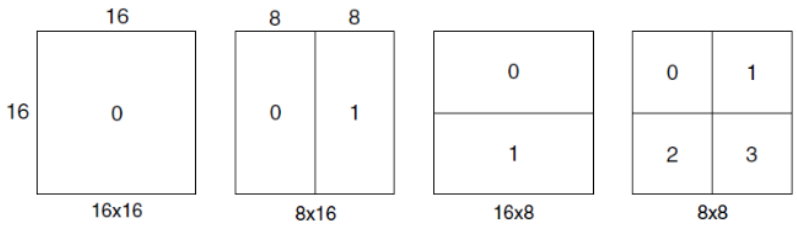



Figure 3.3: *Macroblock partitions: 16x16, 8x16, 16x8, 8x8*

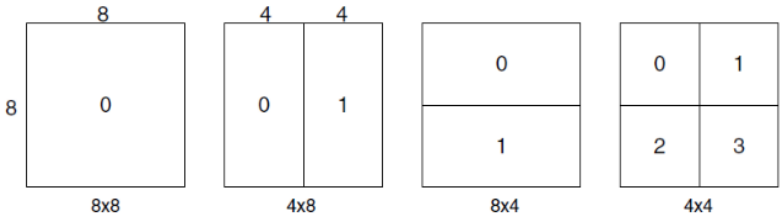



Figure 3.4: *Sub-macroblock partitions: 8x8, 4x8, 8x4, 4x4*

## 3.3 Macroblock Prediction

Most of the macroblocks in H.264/AVC is predicted from previously encoded macroblocks of same slice or slices of different pictures. In the encoder a prediction macroblock is constructed using some dependency information. This dependency information is then passed to the decoder and the decoder generates the same predicted macroblock using this information. As much the prediction is correct, the residual information in the encoded data is more less. According to the dependency information needed to decode a macroblock, this prediction can be of two types.

**Intra Prediction:** In intra prediction mode, a macroblock is predicted using the information from neighbouring macroblocks of the same slice, which are already decoded. For a macroblock, the neighbouring macroblocks on which it may depend are shown in figure 3.5. In this figure, information from macroblock A, B, C and D may be used to decode the current macroblock. H.264/AVC have several modes of prediction for luma and chroma samples of a macroblock described in chapter 4.

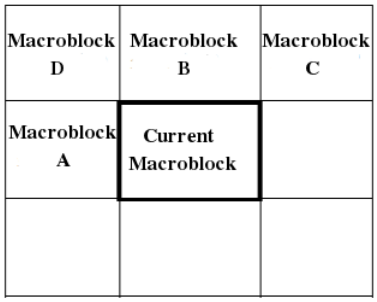



Figure 3.5: *Neighbouring Macroblocks for a given macroblock*

**Inter Prediction:** In inter prediction, a macroblock is predicted from macroblocks of one or more already decoded frames or fields using block based motion compensation. In H.264/AVC the block sizes varies from 16x16 to 4x4 as shown in figure 3.3 and figure 3.4. The encoded bit-stream contains the choice of partitions of a macroblock. To predict each partition it needs one or two motion vectors. Motion vectors are predicted from the motion vectors of already decoded neighbouring partitions. Encoded data contains the difference between original motion vector and predicted motion vector, this difference is added to the predicted motion vector in decoder to get the actual motion vector.
For current macroblock E the choice of partitions from which the motion vector is predicted are shown in figure 3.6 and figure 3.7 if they are available.

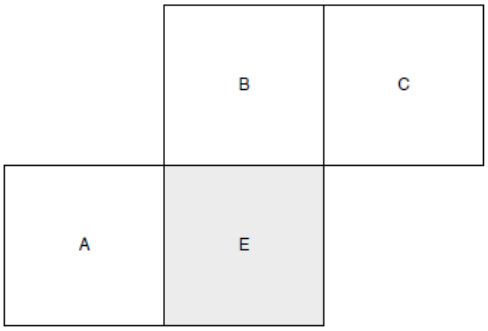



Figure 3.6: *Neighbouring partitions for motion vector prediction when partition sizes are same*

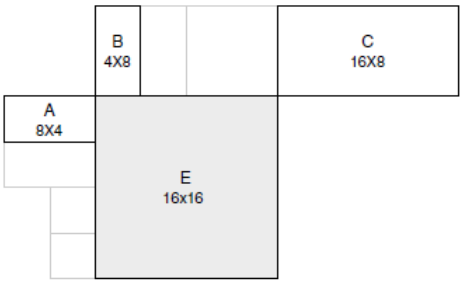



Figure 3.7: *Neighbouring partitions for motion vector prediction when partition sizes are different*

The motion vectors for partition E is predicted inthe following way when partition A, B and C are available.

1. For transmitted partitions excluding 16x8 and 8x16 partition sizes, motion vector is predicted by the median of the motion vectors for partitions A, B and C.

2. For 16x8 partitions, motion vector for the upper 16x8 partition is predicted from B and motion vector for the lower 16x8 partition is predicted from A.

3. For 8x16 partitions,motion vector for the left 8x16 partition is predicted from A and motion vector for the right 8x16 partition is predicted from C.

## 3.4 Macroblock Level Parallelism

To parallelize the decoding task of a H.264/AVC decoder, we have to divide and distribute the video data among several cores. This division can be done in various levels,

1. Group of pictures level

2. Frame level

3. Slice level

4. Macroblock level

From previous studies on H.264/AVC parallelization, we have observed that sufficient scalability can not be achieved through the other methods except macroblock level parallelism. In H.264/AVC usually MBs in a frame are processed in scan order, which means starting from the top left corner of the frame and moving to the right, row after row. In macroblock level parallelism, macroblocks are processed in parallel considering the data dependencies. But the challenges in this method are,

**For I-Slices:** There are a large amount of spatial data dependencies in parallel macroblock decoding for an I-Slice. Each macroblock is predicted from its neighboring macroblocks, which may be decoded in another core. Thus we have to generate a suitable scheduling algorithm so that this data dependency among cores will be minimized. This is true for motion vector prediction for P and B slices as well.

**For P and B-Slices:** To decode macroblocks of P or B-slices, it may refer to a macroblock of different frame or field which is in decoding process in another core. So it has to wait till the decoding of all the slices of that frame or field is completed. This problem can be solved if we can make the proper macroblock available for reference when its decoding is completed. Then the current macroblock can be processed before the completion of its reference picture.

A few other relevant issues applicable to the parallelization approach are as follows:

1. Maintain functional correctness of program.
2. Support macroblock adaptive frame field coding.
3. Measure of multicore utilization (speed gain over single core).
4. Latency impact on display of the pictures must not hamper.
5. Have good efficiency on defined test set of sequences of common resolutions CIF, D1, 720, 1080 and higher.

In this work, we investigate the performance scalability of Macroblock-level parallelization for the H.264 decoder in a multi-core environment where multiple processing cores, each having a fixed amount of small local cache and having access to a larger shared cache, are placed on the same chip. Specifically, our focus is on reducing the access of shared caches and inter- core communication by increasing the level of reuse for local caches, while ensuring the dependency constraints of a given workload. In multicore platforms, reducing shared cache access can result in decreased execution times, which may allow a larger workload to be supported or hardware requirements (or costs) to be reduced.

Given a system of N cores, each with a fixed amount of local memory and each having access to a shared memory, our objectives are to develop a scheduling and allocation strategy for scheduling multiple macroblocks with a target of minimizing off-chip access, while satisfying dependency constraints. Our model has the following assumptions.

### 3.4.1 Assumptions:

1. Entropy decoding is decoupled from the parallelized decoder before the allocation of macroblocks are started.
2. Incoming video stream is pre-parsed and all slices extracted and stored in a buffer.
3. Message passing time between processors ignored
4. Deblocking filter will be used on a picture after completion of all the prefiltering subtasks on all the macroblocks of the picture.

# Chapter 4

# Dependence Study for Intra Prediction

Luma and Chroma components of a Macroblock are predicted separately as specified in encoded data for that macroblock. We are interested in the amount of data required from the neighbouring macroblocks to predict the current macroblock.

## 4.1 Intra Prediction for Luma Samples

Prediction of luma blocks in H.264/AVC can be done in three ways. A 16x16 luma block of a macroblock can be predicted by predicting sixteen 4x4 blocks or four 8x8 blocks or predicting the whole 16x16 block. There are different modes of prediction for each such blocks; prediction mode of a block is derived from the previously derived prediction mode of adjacent blocks and their availability.

### 4.1.1 Intra 4x4 Prediction for Luma Samples:

There can be nine different modes of prediction for each of the sixteen 4x4 luma blocks. The following table 4.1 specifies the value and corresponding name of these modes along-with the data dependency.

For a 4x4 block like Figure 4.1, a subset of neighbouring 13 samples are used to predict the block depending on the prediction mode of the block. For a particular prediction mode, the number of samples required from side W, X, Y and Z of the 4x4 block are also given in table 4.1. In figure 4.2 the direction of these dependency for each prediction modes are also given.

| **Intra 4x4 Prediction Mode** | **Name of the Prediction mode** | **Required neighbouring samples** | | | |
|---|---|---|---|---|---|
| | | side W | side X | side Y | side Z |
| 0 | Intra 4x4 Vertical | 0 | 4 | 0 | 0 |
| 1 | Intra 4x4 Horizontal | 4 | 0 | 0 | 0 |
| 2 | Intra 4x4 DC | 4 | 4 | 0 | 0 |
| 3 | Intra 4x4 Diagonal Down Left | 0 | 4 | 4 | 0 |
| 4 | Intra 4x4 Diagonal Down Right | 4 | 4 | 0 | 1 |
| 5 | Intra 4x4 Vertical Right | 4 | 4 | 0 | 1 |
| 6 | Intra 4x4 Horizontal Down | 4 | 4 | 0 | 1 |
| 7 | Intra 4x4 Vertical Left | 0 | 4 | 4 | 0 |
| 8 | Intra 4x4 Horizontal Up | 4 | 0 | 0 | 0 |

Table 4.1: Intra 4x4 Prediction Modes

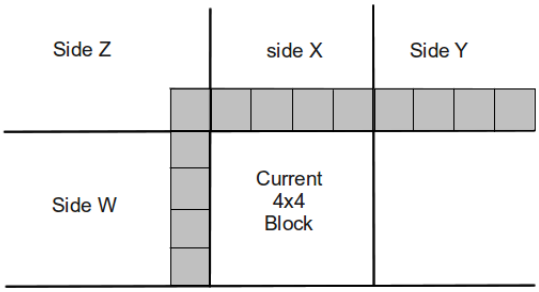



Figure 4.1: *A 4x4 Block with neighbouring 13 samples*

We can divide a macroblock into sixteen 4x4 block, each of which have an intra 4x4 prediction mode for luma samples. As in figure 4.3, the top and left boundary blocks M, N, O, P, Q, R, S are responsible for data dependency from previously decoded macroblocks.
Thus there are $9^7 = 4782969$ possible combinations of prediction modes and for each such combinations there will be a data requirement from neighbouring macroblocks. With respect to the amount of data required from neighbouring

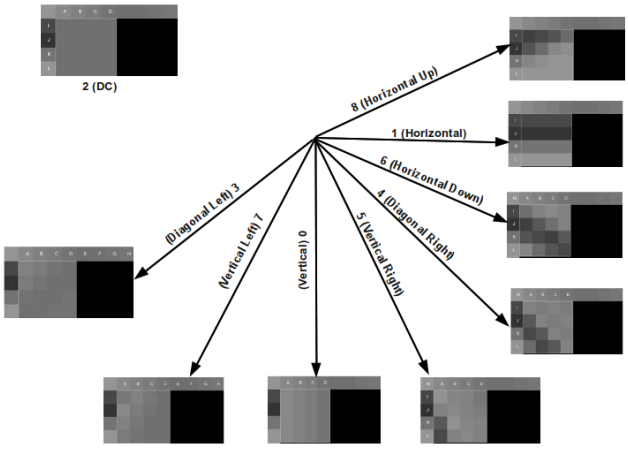



Figure 4.2: *Direction of dependency for different prediction modes of Intra 4x4 Prediction for luma samples*

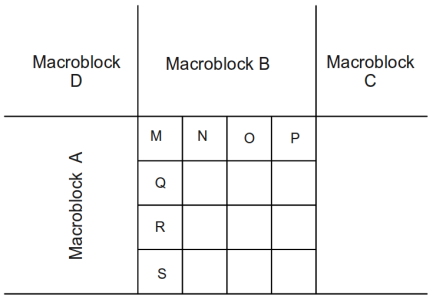



Figure 4.3: *Macroblock divided into sixteen 4x4 block*

macroblocks A, B, C and D, we can classify the current macroblock. There are 180 possible combinations of amount of data requirements from neighbouring macroblocks given in table 4.2

| Data from | | | | Data from | | | | Data from | | | | Data from | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | A | B | C | D | A | B | C | D | A | B | C | D |
| 0 | 16 | 0 | 0 | 12 | 13 | 0 | 0 | 13 | 0 | 0 | 0 | 12 | 16 | 4 | 1 |
| 4 | 16 | 0 | 0 | 10 | 13 | 0 | 0 | 16 | 0 | 0 | 0 | 13 | 16 | 4 | 1 |
| 5 | 16 | 0 | 0 | 13 | 13 | 0 | 0 | 4 | 4 | 4 | 0 | 16 | 16 | 4 | 1 |
| 8 | 16 | 0 | 0 | 0 | 4 | 0 | 0 | 8 | 4 | 4 | 0 | 4 | 8 | 0 | 1 |
| 9 | 16 | 0 | 0 | 4 | 4 | 0 | 0 | 9 | 4 | 4 | 0 | 8 | 8 | 0 | 1 |
| 12 | 16 | 0 | 0 | 5 | 4 | 0 | 0 | 12 | 4 | 4 | 0 | 9 | 8 | 0 | 1 |
| 10 | 16 | 0 | 0 | 8 | 4 | 0 | 0 | 13 | 4 | 4 | 0 | 12 | 8 | 0 | 1 |
| 13 | 16 | 0 | 0 | 9 | 4 | 0 | 0 | 16 | 4 | 4 | 0 | 13 | 8 | 0 | 1 |
| 0 | 12 | 0 | 0 | 12 | 4 | 0 | 0 | 4 | 5 | 0 | 0 | 16 | 8 | 0 | 1 |
| 4 | 12 | 0 | 0 | 10 | 4 | 0 | 0 | 8 | 5 | 0 | 0 | 4 | 12 | 4 | 1 |
| 5 | 12 | 0 | 0 | 13 | 4 | 0 | 0 | 9 | 5 | 0 | 0 | 8 | 12 | 4 | 1 |
| 8 | 12 | 0 | 0 | 0 | 8 | 4 | 0 | 12 | 5 | 0 | 0 | 9 | 12 | 4 | 1 |
| 9 | 12 | 0 | 0 | 4 | 8 | 4 | 0 | 13 | 5 | 0 | 0 | 12 | 12 | 4 | 1 |
| 12 | 12 | 0 | 0 | 5 | 8 | 4 | 0 | 16 | 5 | 0 | 0 | 13 | 12 | 4 | 1 |
| 10 | 12 | 0 | 0 | 8 | 8 | 4 | 0 | 4 | 9 | 4 | 0 | 16 | 12 | 4 | 1 |
| 13 | 12 | 0 | 0 | 9 | 8 | 4 | 0 | 8 | 9 | 4 | 0 | 4 | 13 | 0 | 1 |
| 0 | 16 | 4 | 0 | 12 | 8 | 4 | 0 | 9 | 9 | 4 | 0 | 8 | 13 | 0 | 1 |
| 4 | 16 | 4 | 0 | 10 | 8 | 4 | 0 | 12 | 9 | 4 | 0 | 9 | 13 | 0 | 1 |
| 5 | 16 | 4 | 0 | 13 | 8 | 4 | 0 | 13 | 9 | 4 | 0 | 12 | 13 | 0 | 1 |
| 8 | 16 | 4 | 0 | 0 | 9 | 0 | 0 | 16 | 9 | 4 | 0 | 13 | 13 | 0 | 1 |
| 9 | 16 | 4 | 0 | 4 | 9 | 0 | 0 | 16 | 13 | 0 | 0 | 16 | 13 | 0 | 1 |
| 12 | 16 | 4 | 0 | 5 | 9 | 0 | 0 | 16 | 13 | 4 | 0 | 4 | 4 | 0 | 1 |
| 10 | 16 | 4 | 0 | 8 | 9 | 0 | 0 | 4 | 10 | 0 | 0 | 8 | 4 | 0 | 1 |
| 13 | 16 | 4 | 0 | 9 | 9 | 0 | 0 | 8 | 10 | 0 | 0 | 9 | 4 | 0 | 1 |
| 0 | 8 | 0 | 0 | 12 | 9 | 0 | 0 | 9 | 10 | 0 | 0 | 12 | 4 | 0 | 1 |
| 4 | 8 | 0 | 0 | 10 | 9 | 0 | 0 | 12 | 10 | 0 | 0 | 13 | 4 | 0 | 1 |
| 5 | 8 | 0 | 0 | 13 | 9 | 0 | 0 | 13 | 10 | 0 | 0 | 16 | 4 | 0 | 1 |
| 8 | 8 | 0 | 0 | 0 | 13 | 4 | 0 | 16 | 10 | 0 | 0 | 4 | 8 | 4 | 1 |
| 9 | 8 | 0 | 0 | 4 | 13 | 4 | 0 | 16 | 16 | 0 | 0 | 8 | 8 | 4 | 1 |
| 12 | 8 | 0 | 0 | 5 | 13 | 4 | 0 | 16 | 16 | 4 | 0 | 9 | 8 | 4 | 1 |
| 10 | 8 | 0 | 0 | 8 | 13 | 4 | 0 | 4 | 16 | 0 | 1 | 12 | 8 | 4 | 1 |
| 13 | 8 | 0 | 0 | 9 | 13 | 4 | 0 | 8 | 16 | 0 | 1 | 13 | 8 | 4 | 1 |
| 0 | 12 | 4 | 0 | 12 | 13 | 4 | 0 | 9 | 16 | 0 | 1 | 16 | 8 | 4 | 1 |
| 4 | 12 | 4 | 0 | 10 | 13 | 4 | 0 | 12 | 16 | 0 | 1 | 4 | 9 | 0 | 1 |
| 5 | 12 | 4 | 0 | 13 | 13 | 4 | 0 | 13 | 16 | 0 | 1 | 8 | 9 | 0 | 1 |
| 8 | 12 | 4 | 0 | 16 | 12 | 0 | 0 | 16 | 16 | 0 | 1 | 9 | 9 | 0 | 1 |
| 9 | 12 | 4 | 0 | 16 | 8 | 0 | 0 | 4 | 12 | 0 | 1 | 12 | 9 | 0 | 1 |
| 12 | 12 | 4 | 0 | 16 | 12 | 4 | 0 | 8 | 12 | 0 | 1 | 13 | 9 | 0 | 1 |
| 10 | 12 | 4 | 0 | 16 | 4 | 0 | 0 | 9 | 12 | 0 | 1 | 16 | 9 | 0 | 1 |
| 13 | 12 | 4 | 0 | 16 | 8 | 4 | 0 | 12 | 12 | 0 | 1 | 4 | 13 | 4 | 1 |
| 0 | 13 | 0 | 0 | 16 | 9 | 0 | 0 | 13 | 12 | 0 | 1 | 8 | 13 | 4 | 1 |
| 4 | 13 | 0 | 0 | 4 | 0 | 0 | 0 | 16 | 12 | 0 | 1 | 9 | 13 | 4 | 1 |
| 5 | 13 | 0 | 0 | 8 | 0 | 0 | 0 | 4 | 16 | 4 | 1 | 12 | 13 | 4 | 1 |
| 8 | 13 | 0 | 0 | 9 | 0 | 0 | 0 | 8 | 16 | 4 | 1 | 13 | 13 | 4 | 1 |
| 9 | 13 | 0 | 0 | 12 | 0 | 0 | 0 | 9 | 16 | 4 | 1 | 16 | 13 | 4 | 1 |

Table 4.2:

### 4.1.2 Intra 8x8 Prediction for Luma Samples:

Similarly luma components of a macroblock can be predicted by predicting four 8x8 block. For luma 8x8 block prediction also, there are nine prediction modes. In the table 4.3 the prediction modes along with their name and the dependency from neighbouring 25 samples are given. Direction of dependency for these prediction modes are same as 4x4 prediction modes shown in figure 4.2.

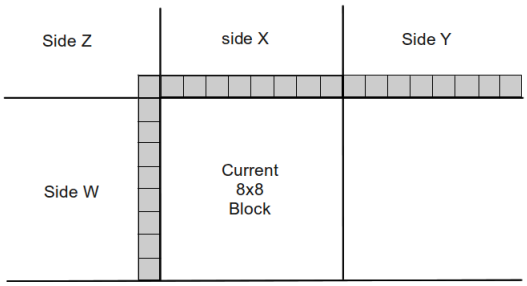



Figure 4.4: *A 8x8 Block with neighbouring 25 samples*

| **Intra 8x8 Prediction Mode** | **Name of the Prediction mode** | **Required neighbouring samples** | | | |
|---|---|---|---|---|---|
| | | side W | side X | side Y | side Z |
| 0 | Intra 8x8 Vertical | 0 | 8 | 0 | 0 |
| 1 | Intra 8x8 Horizontal | 8 | 0 | 0 | 0 |
| 2 | Intra 8x8 DC | 8 | 8 | 0 | 0 |
| 3 | Intra 8x8 Diagonal Down Left | 0 | 8 | 8 | 0 |
| 4 | Intra 8x8 Diagonal Down Right | 8 | 8 | 0 | 1 |
| 5 | Intra 8x8 Vertical Right | 8 | 8 | 0 | 1 |
| 6 | Intra 8x8 Horizontal Down | 8 | 8 | 0 | 1 |
| 7 | Intra 8x8 Vertical Left | 0 | 8 | 8 | 0 |
| 8 | Intra 8x8 Horizontal Up | 8 | 0 | 0 | 0 |

Table 4.3: Intra 8x8 Prediction Modes

For a 16x16 macroblock, its three boundary 8x8 block M, N, O is responsible for intra data dependency as shown in figure 4.5. We can again classify the $9^3 = 243$ combinations of prediction modes that is possible in a macroblock, so that in each class amount of data needed from macroblock A, B, C and D to decode the current macroblock becomes same. For each such class, the data requirements are given in table 4.4.

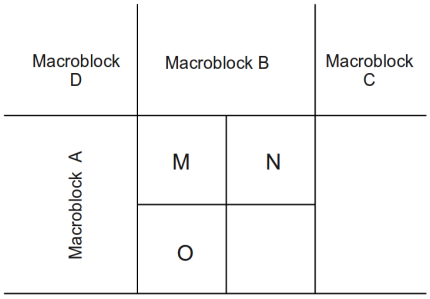



Figure 4.5: *Macroblock divided into four 8x8 block*

| **Data from** | | | | **Data from** | | | | **Data from** | | | | **Data from** | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | A | B | C | D | A | B | C | D | A | B | C | D |
| 0 | 16 | 0 | 0 | 0 | 16 | 8 | 0 | 8 | 8 | 8 | 0 | 8 | 16 | 0 | 1 |
| 8 | 16 | 0 | 0 | 8 | 16 | 8 | 0 | 16 | 8 | 8 | 0 | 16 | 16 | 0 | 1 |
| 9 | 16 | 0 | 0 | 9 | 16 | 8 | 0 | 8 | 9 | 0 | 0 | 8 | 8 | 0 | 1 |
| 0 | 8 | 0 | 0 | 16 | 8 | 0 | 0 | 16 | 9 | 0 | 0 | 16 | 8 | 0 | 1 |
| 8 | 8 | 0 | 0 | 8 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 8 | 16 | 8 | 1 |
| 9 | 8 | 0 | 0 | 16 | 0 | 0 | 0 | 16 | 16 | 8 | 0 | 16 | 16 | 8 | 1 |

Table 4.4:

### 4.1.3 Intra 16x16 Prediction for Luma Samples:

A macroblock can be predicted as a single 16x16 block. There are four prediction modes for luma 16x16 block intra prediction as shown in table 4.5 alongwith the data dependency from neighbouring macroblocks.

| **Intra 16x16 Prediction Mode** | **Name of the Prediction mode** | **Required samples from** | | | |
|---|---|---|---|---|---|
| | | mb A | mb B | mb C | mb D |
| 0 | Intra 16x16 Vertical | 0 | 16 | 0 | 0 |
| 1 | Intra 16x16 Horizontal | 16 | 0 | 0 | 0 |
| 2 | Intra 16x16 DC | 16 | 16 | 0 | 0 |
| 3 | Intra 16x16 Plane | 16 | 16 | 0 | 1 |

Table 4.5: Intra 16x16 Prediction Modes

## 4.2 Intra Prediction for Chroma samples

For different chroma array type, the prediction process for chroma samples are different.

### For Chroma Array Type = 1

The following table shows the no of distinct possible combination of chroma pixel data that a macroblock has to read from its neighboring macroblocks when Chroma Array Type is 1.

| **Intra 16x16 Prediction Mode** | **Name of the Prediction mode** | **Required samples from** | | | |
|---|---|---|---|---|---|
| | | mb A | mb B | mb C | mb D |
| 0 | Intra Chroma DC | 8 | 8 | 0 | 0 |
| 1 | Intra Chroma Vertical | 8 | 0 | 0 | 0 |
| 2 | Intra1 Chroma Horizontal | 0 | 8 | 0 | 0 |
| 3 | Intra Chroma Plane | 8 | 8 | 0 | 1 |

Table 4.6: Intra Chroma Prediction Modes for Chroma array Type = 1

### For Chroma Array Type = 2

The following table shows the no of distinct possible combination of chroma pixel data that a macroblock has to read from its neighboring macroblocks when Chroma Array Type is 2.

| **Intra 16x16 Prediction Mode** | **Name of the Prediction mode** | **Required samples from** | | | |
|---|---|---|---|---|---|
| | | mb A | mb B | mb C | mb D |
| 0 | Intra Chroma DC | 16 | 8 | 0 | 0 |
| 1 | Intra Chroma Vertical | 16 | 0 | 0 | 0 |
| 2 | Intra1 Chroma Horizontal | 0 | 8 | 0 | 0 |
| 3 | Intra Chroma Plane | 16 | 8 | 0 | 1 |

Table 4.7: Intra Chroma Prediction Modes for Chroma Array Type = 2

# Chapter 5

# Cache Profiling

As our objective is to increase the performance of the decoder, we have checked how it works on different cache environment. We can reduce the number of data read and write misses by increasing the associativity in different level of cache. For different associativities of first level instruction cache, first level data cache and third level unified cache, the cache miss penalty of a H.264 decoder is given in following two tables.

In all the cases,
L1 Instruction Cache: size = 32768 Byte and line size = 64 Byte
L1 Data Cache: size = 32768 Byte and line size = 64 Byte
L3 Cache: size = 3145728 Byte and line size = 64 Byte
Assuming penalty for single L1 cache miss is 10 cycles and for single L3 cache miss it is 200 cycles.

Executing the decoder on a set of video streams with display resolution 176x144 and frame rate 25fps, cache miss penalties are given in table 5.1 and table 5.2 is for a set of video streams with display resolution 1280x720 and frame rate of 25fps.In both the tables, second column is for 4 way associative L1 instruction cache, 4 way associative L1 data cache and 12 way associativie L3 cache; third column is for 4 way associative L1 instruction cache, 8 way associative L1 data cache and 12 way associativie L3 cache; fourth column is for 8 way associative L1 instruction cache, 4 way associative L1 data cache and 12 way associativie L3 cache; fifth column is for 8 way associative L1 instruction cache, 8 way associative L1 data cache and 12 way associativie L3 cache. The corresponding graphs are given in figure 5.1 and figure 5.2.

| Video streams | I1 = 4, D1 = 4, L3 = 12 | I1 = 4, D1 = 8, L3 = 12 | I1 = 8, D1 = 4, L3 = 12 | I1 = 8, D1 = 8, L3 = 12 |
|---|---|---|---|---|
| 1 | 11964300 | 11938660 | 11158930 | 11133290 |
| 3 | 11139540 | 11076810 | 10429140 | 10366410 |
| 4 | 15053450 | 15012890 | 14724550 | 14683990 |
| 5 | 12595410 | 12574370 | 11367560 | 11347040 |
| 6 | 13732080 | 13719940 | 13419870 | 13407730 |
| 7 | 12667270 | 12633100 | 12185700 | 12151530 |
| 8 | 12678140 | 12634990 | 12262890 | 12219740 |
| 9 | 12842460 | 12815610 | 12451250 | 12424400 |
| 10 | 12824060 | 12780570 | 12378140 | 12334650 |
| 11 | 12865390 | 12817250 | 12411550 | 12363410 |
| 12 | 12686340 | 12637320 | 12167820 | 12118800 |
| 13 | 11805420 | 11622600 | 10481870 | 10299050 |
| 14 | 11596320 | 11596400 | 10275270 | 10274810 |
| 15 | 12002910 | 12006310 | 10688700 | 10692100 |
| 16 | 11944890 | 11854180 | 11269220 | 11178510 |
| 17 | 12582850 | 12545790 | 12001550 | 11964490 |
| 18 | 11151460 | 11086360 | 10443220 | 10378120 |

Table 5.1: Cache miss penalty of the decoder for different associativities of cache, when applied on video streams of display resolution 176x144 and 25 frame/sec

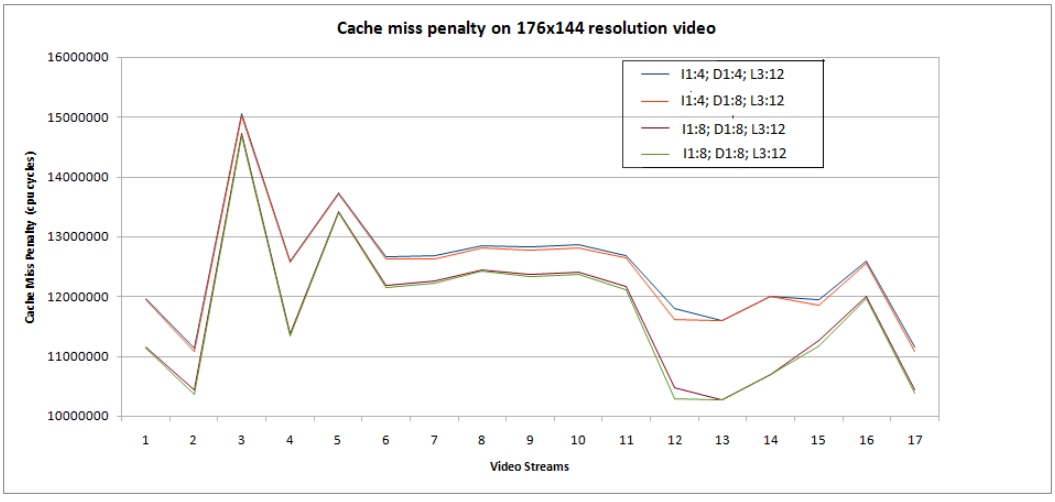



Figure 5.1: *Graph representing Cachemisspenalty to decode coded video streams of resolution 176x144*

| Video streams | I1 = 4, D1 = 4, L3 = 12 | I1 = 4, D1 = 8, L3 = 12 | I1 = 8, D1 = 4, L3 = 12 | I1 = 8, D1 = 8, L3 = 12 |
|---|---|---|---|---|
| 1 | 7661572280 | 7660185200 | 7255868240 | 7254480970 |
| 2 | 6920854200 | 6919621180 | 6678286320 | 6677048170 |
| 3 | 7901699280 | 7894705020 | 7510526070 | 7503532760 |
| 4 | 8012060920 | 8008696830 | 7539654140 | 7536289110 |
| 5 | 7372048390 | 7365453420 | 7103300690 | 7096704390 |
| 6 | 7493155880 | 7491491420 | 7159584670 | 7157919070 |
| 7 | 7322999920 | 7321467770 | 6999490020 | 6997957870 |
| 8 | 8122050320 | 8118966680 | 7720041070 | 7716956670 |
| 9 | 8019448800 | 8016592940 | 7653015840 | 7650158460 |
| 10 | 7534994860 | 7532989410 | 7179414970 | 7177404010 |
| 11 | 6676457940 | 6675666480 | 6445371250 | 6444579600 |
| 12 | 7437291830 | 7435717550 | 7108998220 | 7107423940 |

Table 5.2: Cache miss penalty of the decoder for different associativities of cache when applied on video streams with display resolution 1280x720 and frame rate of 25fps

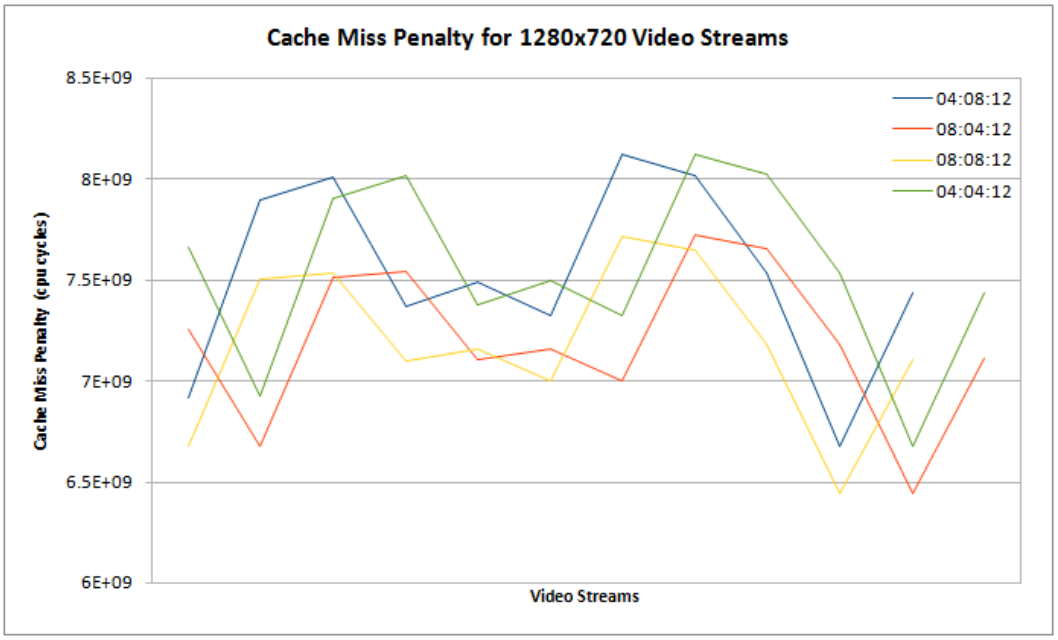



Figure 5.2: *Graph representing Cachemisspenalty to decode coded video streams of resolution 1280x720(04:04:12 represents associativity of I1 cache is 4; D1 cache is 4 and for L3 cache associativity is 12)*

# Chapter 6

# ILP Formulation

## 6.1 Notation

Suppose we have a $m \times n$ slice of macroblocks, tasks are denoted as:

| | | | | |
|---|---|---|---|---|
| $t_1$ | $t_2$ | $t_3$ | ... | $t_n$ |
| $t_{n+1}$ | $t_{n+2}$ | $t_{n+3}$ | ... | $t_{2n}$ |
| ... | ... | ... | ... | ... |
| $t_{(m-1)n+1}$ | $t_{(m-1)n+2}$ | $t_{(m-1)n+3}$ | ... | $t_{mn}$ |

## 6.2 ILP Formulation I

Suppose we have $n$ macroblocks in a slice. We want to schedule the decoding task of these $n$ macroblocks into $r$ processors.The data dependencies between the macroblocks are given by the dependency graph $G = (T, E)$.

### 6.2.1 Variables:

$T = \{t_1, t_2, ..., t_n\}$ denotes the set of tasks.
$E = \{(t_i, t_j)|$ if $t_i, t_j \in T$ and $t_j$ depends on $t_i\}$
$D(j) = \{t_i \in T|$ if $t_i, t_j \in T$ and $(t_i, t_j) \in E\}$
$P = \{p_1, p_2, ..., p_r\}$ denotes the set of processors.

for each task $t_i \in T$,
i) Let $st_i$ be the start time of execution of task $t_i$.
ii) Let $l_i$ be the task length of task $t_i$ in CPU cycles.

$\forall t_i \in T, p_k \in P$ and $1 \leq s \leq n$

$$y_{ik}^s = \begin{cases} 1 & \text{if task } t_i \text{ be the } s^{th} \text{ task on processor } p_k \\ 0 & \text{otherwise} \end{cases}$$

$\forall t_i, t_j \in T$
Let $c_{ij}$ = amount of data from task $t_i$ needed to decode task $t_j$

$\forall p_h, p_k \in P$
Let $d_{hk}$ be the CPU cycles needed to transfer unit pixel data from $p_h$ to $p_k$

$\forall t_i, t_j \in T$ and $\forall p_h, p_k \in P$
Let communication latency $\gamma_{ij}^{hk} = c_{ij}\, d_{hk}$

Let $MAX$ be a constant with very high value.

### 6.2.2 Objective:

$$minimize \qquad \max_{t_i \in T} \{st_i + l_i\} \tag{6.2.2.0.1}$$

### 6.2.3 Constraints:

- Each task will be executed on a single processor and only once, $\forall t_i \in T$ :

$$\sum_{p_k \in P} \sum_{s=1}^{n} y_{ik}^{s} = 1 \tag{6.2.3.0.2}$$

  If there is four tasks to be allocated on two processors, then for the first task,
  $y_{1,1}^1 + y_{1,1}^2 + y_{1,1}^3 + y_{1,1}^4 + y_{1,2}^1 + y_{1,2}^2 + y_{1,2}^3 + y_{1,2}^4 = 1$

- At most one task will be the first task on a processor, $\forall p_k \in P$ :

$$\sum_{t_i \in T} y_{ik}^{1} \leq 1 \tag{6.2.3.0.3}$$

  If there is four tasks to be allocated on two processors, then for the first processor,
  $y_{1,1}^1 + y_{2,1}^1 + y_{3,1}^1 + y_{4,1}^1 = 1$

- $\forall (s \geq 2)$ if some task is the $s^{th}$ task assigned to a processor, then there must be another $(s-1)^{th}$ task assigned on that processor, $\forall p_k \in P$ and $2 \leq s \leq n$ :

$$\sum_{t_i \in T} y_{ik}^{s} \leq \sum_{t_i \in T} y_{ik}^{s-1} \tag{6.2.3.0.4}$$

  If there is four tasks to be allocated on two processors, then for the first processor when $s = 3$,
  $y_{1,1}^3 + y_{2,1}^3 + y_{3,1}^3 + y_{4,1}^3 \leq y_{1,1}^2 + y_{2,1}^2 + y_{3,1}^2 + y_{4,1}^2$

- A task can start its execution if and only if all the task its depends on are finished and required data is transferred, $\forall t_j \in T$ and $t_i \in D(j)$ :

$$st_j \geq st_i + l_i + \sum_{p_h \in P} \sum_{s=1}^{n} \sum_{p_k \in P} \sum_{r=1}^{n} \gamma_{ij}^{hk} y_{ih}^{s} y_{jk}^{r} \tag{6.2.3.0.5}$$

After linearization this constraint can be replaced with a set of constraints,
$st_j \geq st_i + l_i + \sum_{p_h \in P} \sum_{s=1}^{n} \sum_{p_k \in P} \sum_{r=1}^{n} \gamma_{ij}^{hk} q_{ijhk}^{sr}$
and $\forall p_h,\ p_k \in P;\ s,\ r \in \{1, 2, ..., n\}$
$q_{ijhk}^{sr} \leq y_{ih}^{s}$
$q_{ijhk}^{sr} \leq y_{jk}^{r}$
$q_{ijhk}^{sr} \geq y_{ih}^{s} + y_{jk}^{r} - 1$
$q_{ijhk}^{sr} \geq 0$

If there is three tasks to be allocated on two processors, and task $t_2$ is depend on task $t_1$then,

$st_2 \geq st_1 + l_1 + \gamma_{1,2}^{1,1} q_{1,2,1,1}^{1,1} + \gamma_{1,2}^{1,1} q_{1,2,1,1}^{1,2} + \gamma_{1,2}^{1,1} q_{1,2,1,1}^{1,3} + \gamma_{1,2}^{1,2} q_{1,2,1,2}^{1,1} + \gamma_{1,2}^{1,2} q_{1,2,1,2}^{1,2} + \gamma_{1,2}^{1,2} q_{1,2,1,2}^{1,3} + \gamma_{1,2}^{1,1} q_{1,2,1,1}^{2,1} + \gamma_{1,2}^{1,1} q_{1,2,1,1}^{2,2} + \gamma_{1,2}^{1,1} q_{1,2,1,1}^{2,3} + \gamma_{1,2}^{1,2} q_{1,2,1,2}^{2,1} + \gamma_{1,2}^{1,2} q_{1,2,1,2}^{2,2} + \gamma_{1,2}^{1,2} q_{1,2,1,2}^{2,3} + \gamma_{1,2}^{2,1} q_{1,2,2,1}^{1,1} + \gamma_{1,2}^{2,1} q_{1,2,2,1}^{1,2} + \gamma_{1,2}^{2,1} q_{1,2,2,1}^{1,3} + \gamma_{1,2}^{2,2} q_{1,2,2,2}^{1,1} + \gamma_{1,2}^{2,2} q_{1,2,2,2}^{1,2} + \gamma_{1,2}^{2,2} q_{1,2,2,2}^{1,3} + \gamma_{1,2}^{2,1} q_{1,2,2,1}^{2,1} + \gamma_{1,2}^{2,1} q_{1,2,2,1}^{2,2} + \gamma_{1,2}^{2,1} q_{1,2,2,1}^{2,3} + \gamma_{1,2}^{2,2} q_{1,2,2,2}^{2,1} + \gamma_{1,2}^{2,2} q_{1,2,2,2}^{2,2} + \gamma_{1,2}^{2,2} q_{1,2,2,2}^{2,3}$
where all '$\gamma$'s are constant with value depending on the dependency amount and cpu cycles needed to read those data. All '$q$'s can be constrained in above mentioned way.

- $\forall p_k \in P;\ t_i,\ t_j \in T$ and $1 \leq s \leq (n-1)$ :

$$st_j \geq st_i + l_i - MAX\left(2 - \left(y_{ik}^{s} + \sum_{r=(s+1)}^{n} y_{jk}^{r}\right)\right) \tag{6.2.3.0.6}$$

For three tasks to be allocated on two processors, if task $t_1$ and $t_2$ are allocated on different processors then for both $k = 1$ and 2, the constraint satisfies. For processor $p_1$, $y_{1,1}^{s}$ and $y_{2,1}^{s}$ both can not be one for any values of $s$ and $r$ thus the constraint satisfies with a very high value of $MAX$. But if task $t_1$ and $t_2$ are allocated on the same processor, then this constraint confirms one of them must be finished before another.

- $\forall p_k \in P;\ t_i \in T$ and $1 \leq s \leq n$ :

$$y_{ik}^{s} \in \{0,\ 1\} \tag{6.2.3.0.7}$$

- $\forall t_i \in T$ :

$$st_i \geq 0 \tag{6.2.3.0.8}$$

### 6.2.4 Discussion:

Though this formulation gives an optimized solution with good utilization, it has certain limitations. The number of non-linear variables in this formulation is very high, to schedule n tasks on p processors equation (6.2.3.0.5) generates $\mathcal{O}(n^4P^2)$ non-linear variables. After linearising the constraints generated by this equation, the number of linear constraints also becomes very large. It also takes a huge amount of time to get the optimized solution. For a 4x4 slice containing

| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ | $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $t_1$ | 0 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_2$ | 0 | 0 | 0 | 0 | 0 | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_3$ | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_4$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_5$ | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_6$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 0 | 0 | 0 | 0 |
| $t_7$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 |
| $t_8$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 9 | 0 | 0 | 0 | 0 |
| $t_9$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $t_{10}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 0 | 0 | 16 | 0 | 0 |
| $t_{11}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 13 | 1 |
| $t_{12}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 |
| $t_{13}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 0 | 0 |
| $t_{14}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 9 | 0 |
| $t_{15}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 |
| $t_{16}$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table 6.1: Dependency between 16 macroblocks of a 4x4 slice

16 macroblocks and with macroblock dependency given in table 6.1, the number of constraints is 97382 and number of variables is 23057 when the number of available cores for allocation is two. Time taken to solve these constraints is 30100 seconds in a quad core processor with 4GB of RAM. The optimized schedule for the 4x4 slice is given in table 6.2 with scheduler length = 7252 cpu cycles.

| | | | |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Table 6.2: Optimized schedule for the 4x4 slice with macroblock dependency given in table 6.1

## 6.3 ILP Formulation II

The number of constraints and variables generated using the previous formulation is huge. So we will go for another formulation which generates smaller number of constraints and variables and also gives the optimized schedule. Suppose we have $n$ macroblocks in a slice. We want to schedule the decoding task of these $n$ macroblocks into $r$ processors.The data dependencies between the macroblocks are given by the dependency graph $G = (T, E)$.

### 6.3.1 Variables:

$T = \{t_1, t_2, ..., t_n\}$ denotes the set of tasks.
$E = \{(t_i, t_j)|$ if $t_i, t_j \in T$ and $t_j$ depends on $t_i\}$
$D(j) = \{t_i \in T|$ if $t_i, t_j \in T$ and $(t_i, t_j) \in E\}$
$P = \{p_1, p_2, ..., p_r\}$ denotes the set of processors.

for each task $t_i \in T$,
i) Let $st_i$ be the start time of execution of task $t_i$.
ii) Let $l_i$ be the task length of task $t_i$ in CPU cycles.

$\forall t_i \in T$ and $p_k \in P$,

Let $x_{ik} = \begin{cases} 1 & \text{if task } t_i \text{ is scheduled on processor } p_k \\ 0 & \text{otherwise} \end{cases}$

Let $y_{ik} = \begin{cases} 1 & \text{if task } t_i \text{ is scheduled on processor } p_k;\ \exists\ t_j \text{ s.t } t_i \in D(j) \\ & \text{which is scheduled on a different processor other than } p_k \\ 0 & \text{otherwise} \end{cases}$

$\forall t_i,\ t_j \in T$ and $p_k \in P$,

Let $z_{ij}^k = \begin{cases} 1 & \text{if task } t_i,\ t_j \text{ is scheduled on same processor } p_k \\ & \text{and } t_i \text{ starts before } t_j \\ 0 & \text{otherwise} \end{cases}$

$\forall t_i, t_j \in T$,
Let $c_{ij}$ = amount of data from task $t_i$ needed to decode task $t_j$

$\forall p_h, p_k \in P$,
Let $d_{hk}$ be the CPU cycles needed to transfer unit pixel data from $p_h$ to $p_k$

$\forall t_i, t_j \in T$ and $\forall p_h, p_k \in P$,
Let communication latency $\gamma_{ij}^{hk} = c_{ij}\ d_{hk}$

Let $MAX$ be a constant with very high value.

### 6.3.2 Objective:

$$minimize \qquad \max_{t_i \in T} \{st_i + l_i\} \tag{6.3.2.0.9}$$

### 6.3.3 Constraints:

- Each task must be assigned to a single processor, $\forall t_i \in T$ :

$$\sum_{p_k \in P} x_{ik} = 1 \tag{6.3.3.0.10}$$

  If number of available processors is four then for a task $t_1$,
  $x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1$

- A task can be executed if all the tasks it depends on, are finished

  - If both tasks are scheduled on same processor, then $\forall p_k \in P$, $\forall t_j \in T$ and $t_i \in D(j)$ :

$$st_i + l_i + \gamma_{ij}^{kk} \leq st_j + (2 - x_{ik} - x_{jk})\, MAX \tag{6.3.3.0.11}$$

    for a processor $p_1$ and two tasks $t_1$ and $t_2$,where $t_2$ depends on $t_1$, the constraint is,
    $st_1 + l_1 + \gamma_{1,2}^{1,1} \leq st_2 + (2 - x_{1,1} - x_{2,1})\, MAX$
    If both the tasks are allocated on same processor $p_1$, then $x_{1,1} = 1$ and $x_{2,1} = 1$. So the constraint becomes
    $st_1 + l_1 + \gamma_{1,2}^{1,1} \leq st_2$
    which means $t_2$ can be started after $t_1$ finishes and getting required data from L1 cache of processor $p_1$.
    Otherwise if two tasks are allocated on different processors then one or both of $x_{1,1}$ and $x_{2_1}$ becomes zero satisfying the constraint with a very high value of $MAX$.

  - If both tasks are not scheduled on same processor, then $\forall p_h, p_k \in P$, $h \neq k$, $\forall t_j \in T$ and $t_i \in D(j)$ :

$$st_i + l_i + \gamma_{ij}^{hk} \leq st_j + (2 - x_{jk} + x_{ik} - y_{ih})\, MAX \tag{6.3.3.0.12}$$

    for two processors $p_1$, $p_2$ and two tasks $t_1$, $t_2$,where $t_2$ depends on $t_1$, the constraint is
    $st_1 + l_1 + \gamma_{1,2}^{1,2} \leq st_2 + (2 - x_{2,2} + x_{1,2} - y_{1,1})\, MAX$
    if $t_i$ is allocated on $p_1$ and $t_2$ is allocated on $p_2$ then $x_{2,2} = 1$, $x_{1,2} = 0$ and $y_{1,1} = 1$. So the constraint becomes
    $st_1 + l_1 + \gamma_{1,2}^{1,2} \leq st_2$
    which means $t_2$ can be started after $t_1$ finishes and getting required data in $p_2$ from L1 cache of processor $p_1$.
    otherwise, if both tasks allocated in same processor $p_1$ or $p_2$ this constraint satisfies with a very high value of $MAX$.

- Two independent tasks must not be executed on the same processor at the same time, $\forall p_k \in P$ and $\forall t_i, t_j \in T$ where $t_i \notin D(j)$, $t_j \notin D(i)$ :

$$st_i + l_i \leq st_j + (3 - x_{jk} - x_{ik} - z_{ij}^{k})\, MAX \tag{6.3.3.0.13}$$

for a single processor $p_1$ and two independent tasks $t_1$ and $t_2$, the constraint is:
$st_1 + l_1 \leq st_2 + (3 - x_{2,1} - x_{1,1} - z^1_{1,2})\, MAX$
If bothe tasks are allocated on $p_1$ then $x_{1,1} = 1$ and$x_{2,1} = 1$, now if $z^1_{1,2} = 1$ that means task $t_1$ is finished before $t_2$. Otherwise this constraint satisfies with very high value of $MAX$.

$$st_j + l_j \leq st_i + (2 - x_{jk} - x_{ik} + z^k_{ij})\, MAX \tag{6.3.3.0.14}$$

for the same example, the constraint is:
$st_2 + l_2 \leq st_1 + (2 - x_{2,1} - x_{1,1} + z^1_{1,2})\, MAX$
If bothe tasks are allocated on $p_1$ then $x_{1,1} = 1$ and$x_{2,1} = 1$, now if $z^1_{1,2} = 0$ that means task $t_2$ is finished before $t_1$ which implies $z^1_{2,1} = 1$. This constraint confirms that two task can not be overlapped when they are allocated on the same processor. Without the given conditions this constraint satisfies with very high value of $MAX$.

- Start time of each task must be positive, $\forall t_i \in T$ :

$$st_i \geq 0 \tag{6.3.3.0.15}$$

- $\forall t_i \in T$ and $p_k \in P$

$$x_{ik} \in \{0,\, 1\} \tag{6.3.3.0.16}$$

- $\forall t_i \in T$ and $p_k \in P$

$$y_{ik} \in \{0,\, 1\} \tag{6.3.3.0.17}$$

- $\forall t_i,\, t_j \in T$ and $p_k \in P$

$$z^k_{ij} \in \{0,\, 1\} \tag{6.3.3.0.18}$$

### 6.3.4 Discussion:

This formulation generates much less number of variables and constraints than the previous formulation. Time to solve those constraints is also very less.

For the same 4x4 slice to be allocated on two cores and with same macroblock dependency given in table 6.1, the optimized schedule using this formulation is exactly similar to the schedule generated by formulation I. But the number of constraints is only 1168 and number of variables is 515. Time taken to solve these constraints is 129 seconds in a quad core processor with 4GB of RAM. The time it takes to find the optimized schedule is very less in compare to the formulation I, but it is still very slower than the other greedy approaches.

## 6.4 ILP Formulation III

As the previous formulation find an optimized schedule with taking a huge amount of time, we have formulated another ILP which can generate the optimized schedule much faster than the previous formulations.

Suppose we have $n$ macroblocks in a slice. We want to schedule the decoding task of these $n$ macroblocks into $r$ processors.The data dependencies between the macroblocks are given by the dependency graph $G = (T, E)$.

### 6.4.1 Variables:

$T = \{t_1, t_2, ..., t_n\}$ denotes the set of tasks.
$E = \{(t_i, t_j)|$ if $t_i, t_j \in T$ and $t_j$ depends on $t_i\}$
$D(j) = \{t_i \in T|$ if $t_i, t_j \in T$ and $(t_i, t_j) \in E\}$
$P = \{p_1, p_2, ..., p_r\}$ denotes the set of processors.

for each task $t_i \in T$,
i) Let $st_i$ be the start time of execution of task $t_i$.
ii) Let $l_i$ be the task length of task in CPU cycles$t_i$.
iii) Let $pid_i$ be the processor ID where task $t_i$ is to be executed.

Let $w$ be the total execution length of the slice.

$\forall t_i \in T$ and $p_k \in P$,
Let $x_{ik} = \begin{cases} 1 & \text{if task } t_i \text{ is scheduled on processor } p_k \\ 0 & \text{otherwise} \end{cases}$

$\forall t_i, t_j \in T$,
Let $c_{ij}$ = amount of data from task $t_i$ needed to decode task $t_j$
Let $\sigma_{ij} = \begin{cases} 1 & \text{if task } t_i \text{ finishes before } t_j \\ 0 & \text{otherwise} \end{cases}$
Let $\epsilon_{ij} = \begin{cases} 1 & \text{if } pid_i < pid_j \\ 0 & \text{otherwise} \end{cases}$

$\forall p_h, p_k \in P$,
Let $d_{hk}$ be the CPU cycles needed to transfer unit pixel data from $p_h$ to $p_k$

$\forall t_i, t_j \in T$ and $\forall p_h, p_k \in P$,
Let communication latency, $\gamma_{ij}^{hk} = c_{ij}\, d_{hk}$

Let $w$ be the CPU cycles needed to decode the whole slice.

### 6.4.2 Objective:

$$minimize \qquad w \tag{6.4.2.0.19}$$

### 6.4.3 Constraints:

- Each task should be completed before $w, \forall t_i \in T$:

$$st_i + l_i \leq w \tag{6.4.3.0.20}$$

- If there is an ordering between two tasks, then one must be finished before another, $\forall t_i, t_j \in T$ and $t_i \neq t_j$ :

$$st_j \geq st_i + l_i + (\sigma_{ij} - 1)w_{max} \tag{6.4.3.0.21}$$

where $w_{max}$ is defined in (6.4.4.0.37).
For task $t_1$ and $t_2$, if $\sigma_{1,2} = 1$ this constraint says $st_2 \geq st_1 + l_1$ which means task $t_1$ is finished before task $t_2$ starts. And if $\sigma_{1,2} = 0$, this constraint satisfies with a high value of $w_{max}$.

- If two different task allocated on different processor, then one of the processor must have larger id than the other, $\forall t_i, t_j \in T$ and $t_i \neq t_j$ :

$$pid_j \geq pid_i + 1 + (\epsilon_{ij} - 1)|P| \tag{6.4.3.0.22}$$

If $\epsilon_{ij} = 1$ then $pid_i < pid_j$ elase this constraint satisfies.

- Two overlapping tasks must be executed in two different processors and if two tasks are in same processor one of them must be finished before other, $\forall t_i, t_j \in T$ and $t_i \neq t_j$ :

$$\sigma_{ij} + \sigma_{ji} + \epsilon_{ij} + \epsilon_{ji} \geq 1 \tag{6.4.3.0.23}$$

As an example suppose there are two tasks $t_1$ and $t_2$.

$\sigma_{1,2} + \sigma_{2,1} + \epsilon_{1,2} + \epsilon_{2,1} \geq 1$

Condition 1 ($t_1$ and $t_2$ allocated on different processors): then either $pid_1 < pid_2$ or $pid_2 < pid_1$ thus one of $\epsilon_{1,2}$ and $\epsilon_{2,1}$ becomes 1.

Codition 2 ($t_1$ and $t_2$ are two non-overlapping tasks): either $t_1$ finishes before $t_2$ or $t_2$ finishes before $t_1$. So one of $\sigma_{1,2}$ and $\sigma_{2,1}$ becomes 1.

From above we can say, this constraint can not be violated as two overlapping task can not be allocated on the same processor.

- Among two different tasks atmost one can be finished earlier,
$\forall t_i, t_j \in T$ and $t_i \neq t_j$ :

$$\sigma_{ij} + \sigma_{ji} \leq 1 \tag{6.4.3.0.24}$$

For two tasks $t_1$ and $t_2$ this constraint is $\sigma_{1,2} + \sigma_{2,1} \leq 1$. If $t_1$ finishes before $t_2$ then $\sigma_{1,2} = 1$ and $\sigma_{2,1} = 0$. If $t_2$ finishes before $t_1$ then $\sigma_{1,2} = 0$ and $\sigma_{2,1} = 1$. If $t_1$ and $t_2$ are overlapping then $\sigma_{1,2} = 0$ and $\sigma_{2,1} = 0$.

- Among two different tasks atmost one can be allocated to a processor with higher id, $\forall t_i, t_j \in T$ and $t_i \neq t_j$ :

$$\epsilon_{ij} + \epsilon_{ji} \leq 1 \tag{6.4.3.0.25}$$

For two tasks $t_1$ and $t_2$ this constraint is $\epsilon_{1,2} + \epsilon_{2,1} \leq 1$. If $pid_1 < pid_2$ then $\epsilon_{1,2} = 1$ and $\epsilon_{2,1} = 0$. If $pid_1 > pid_2$ then $\epsilon_{1,2} = 0$ and $\epsilon_{2,1} = 1$. If $pid_1 = pid_2$ then $\epsilon_{1,2} = 0$ and $\epsilon_{2,1} = 0$.

- A task can be started only if all the task it depends on are finished. $\forall t_j \in T$ and $\forall t_i \in D(j)$ :

$$\sigma_{ij} = 1 \tag{6.4.3.0.26}$$

If a task $t_a$ depends on three tasks $t_b$, $t_c$ and $t_d$, then execution of $t_a$ can be started only if $t_b$, $t_c$ and $t_d$ finishes. Thus
$\sigma_{ba} = 1$
$\sigma_{ca} = 1$
$\sigma_{da} = 1$

- If a task $t_j$ depends on $t_i$ then $t_j$ can be started after finish time of task $t_i$ added with the communication latency between them, $\forall t_j \in T$ and $\forall t_i \in D(j)$ :

$$st_j \geq st_i + l_i + \sum_{p_h \in P} \sum_{p_k \in P} \gamma_{ij}^{hk} x_{ih} x_{jk} \tag{6.4.3.0.27}$$

After linearization this constraint can be replaced with a set of constraints,
$st_j \geq st_i + l_i + \sum_{p_h \in P} \sum_{p_k \in P} \gamma_{ij}^{hk} q_{ij}^{hk}$
and $\forall p_h$, $p_k \in P$
$q_{ij}^{hk} \leq x_{ih}$
$q_{ij}^{hk} \leq x_{jk}$
$q_{ij}^{hk} \geq x_{ih} + x_{jk} - 1$
$q_{ij}^{hk} \geq 0$

For two tasks $t_1$ and $t_2$ to be allocated on two processors, where $t_2$ depends on $t_1$ the constraint is,
$st_2 \geq st_1 + l_1 + \gamma_{1,2}^{1,1}\, q_{1,2}^{1,1} + \gamma_{1,2}^{1,2}\, q_{1,2}^{1,2} + \gamma_{1,2}^{2,1}\, q_{1,2}^{2,1} + \gamma_{1,2}^{2,2}\, q_{1,2}^{2,2}$

- If task $t_i$ is scheduled on processor $p_k$ then $k$ is processor ID where $t_i$ is to be executed, $\forall t_i \in T$ :

$$\sum_{p_k \in P} k x_{ik} = pid_i \tag{6.4.3.0.28}$$

If number of available processors is four then for a task $t_1$,
$pid_1 = x_{1,1} + 2\, x_{1,2} + 3\, x_{1,3} + 4\, x_{1,4}$
this constraint confirms if $x_{1,j} = 1$, j is the ID of the processor where task $t_1$ is allocated.

- Each task should be assigned to exactly one processor, $\forall t_i \in T$ :

$$\sum_{p_k \in P} x_{ik} = 1 \tag{6.4.3.0.29}$$

If number of available processors is four then for a task $t_1$,
$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} = 1$

- Start time of each task must be non negative, $\forall t_i \in T$ :

$$st_i \geq 0 \tag{6.4.3.0.30}$$

- Total number of CPU cycles needed to decode a slice is non-negative.

$$0 \leq w \tag{6.4.3.0.31}$$

- The allocated processor for a task must be in the set of processors; assuming processors have an unique id,

$$pid_i \in \{1, 2, 3, ..., |P|\} \tag{6.4.3.0.32}$$

- For a task, a processor is either allocated to it or not, $\forall t_i \in T$ and $p_k \in P$ :

$$x_{ik} \in \{0, 1\} \tag{6.4.3.0.33}$$

- $\forall t_i, t_j \in T$:

$$\sigma_{ij} \in \{0, 1\} \tag{6.4.3.0.34}$$

- $\forall t_i, t_j \in T$:

$$\epsilon_{ij} \in \{0, 1\} \tag{6.4.3.0.35}$$

### 6.4.4 Bounds on Objective:

$$w_{min} \leq w \leq w_{max} \tag{6.4.4.0.36}$$

where,

$$w_{max} = \sum_{t_i \in T} l_i + \sum_{t_i, t_j \in T} c_{ij} \max_{p_h, p_k \in P} \{d_{hk}\} \tag{6.4.4.0.37}$$

and

$$w_{min} = \max \left\{ \frac{1}{P} \sum_{i=1}^{n} l_i, \max_{i \leq n} cp(i) \right\} \tag{6.4.4.0.38}$$

$$cp(i) = \begin{cases} l_i & \text{if task } t_i \text{ has no successor} \\ l_i + \max\limits_{i \in D(j)} cp(j) & \text{otherwise} \end{cases}$$

### 6.4.5 Discussion:

This formulation generates a bit higher number of variables and constraints than the formulation II. But as the constraints are very simple, the time to solve those constraints is very less.

On the same slice and processors example, This formulation also gives the same optimized schedule.The number of constraints it generates is 1644 and number of variables is 633. Time taken to solve these constraints is 6 seconds in a quad core processor with 4GB of RAM which is much lesser than the previous two formulations.
The comparison between three ILP formulations is given in table 6.3 taking the same 4x4 slice and scheduling the macroblocks of this slice in 2 cores.

| Formulation no. | No. of Variables | No. of Constraints | time to solve (secs) |
|---|---|---|---|
| I | 23057 | 97382 | 30100 |
| II | 515 | 1168 | 129 |
| III | 633 | 1644 | 6 |

Table 6.3: Comparison Between Three ILP Formulations on an example

### 6.4.6 Overlapped Slice Partitioning:

In all the above mentioned ILP formulations, we are assuming that the decoded unfiltered data of a macroblock will remain in the cache of the processor core on which it was processed; until the complete slice is decoded. But due to limited size of cache memory this may not happen. This limitation does not affect if we apply the formulations on slices with smaller number of macroblocks. Therefore to get a sub-optimal schedule for a larger slice, we can partition it and apply the formulations on them separately. But this is not enough to get the schedule for the whole slice. There may be data dependency between macroblocks of different slice partitions, so starting time of a macroblock in a slice partition may depend on the allocated processor and finishing time of a macroblock which is in a different slice partition. This problem can be solved by overlapping the neighbouring slice partitions and adding some extra constraints with the constraints of formulation III.

As an example, for a slice of size 5x11 we can partition it into two 5x6 slice partition, with one overlapped column of macroblocks between them.

| | | | | | |
|---|---|---|---|---|---|
| $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ |
| $t_{12}$ | $t_{13}$ | $t_{14}$ | $t_{15}$ | $t_{16}$ | $t_{17}$ |
| $t_{23}$ | $t_{24}$ | $t_{25}$ | $t_{26}$ | $t_{27}$ | $t_{28}$ |
| $t_{34}$ | $t_{35}$ | $t_{36}$ | $t_{37}$ | $t_{38}$ | $t_{39}$ |
| $t_{45}$ | $t_{46}$ | $t_{47}$ | $t_{48}$ | $t_{49}$ | $t_{50}$ |

| | | | | | |
|---|---|---|---|---|---|
| $t_6$ | $t_7$ | $t_8$ | $t_9$ | $t_{10}$ | $t_{11}$ |
| $t_{17}$ | $t_{18}$ | $t_{19}$ | $t_{20}$ | $t_{21}$ | $t_{22}$ |
| $t_{28}$ | $t_{29}$ | $t_{30}$ | $t_{31}$ | $t_{32}$ | $t_{33}$ |
| $t_{39}$ | $t_{40}$ | $t_{41}$ | $t_{42}$ | $t_{43}$ | $t_{44}$ |
| $t_{50}$ | $t_{51}$ | $t_{52}$ | $t_{53}$ | $t_{54}$ | $t_{55}$ |

If we see these partitions as two different slices it will be look like,

| | | | | | |
|---|---|---|---|---|---|
| $t_{1,1}$ | $t_{1,2}$ | $t_{1,3}$ | $t_{1,4}$ | $t_{1,5}$ | $t_{1,6}$ |
| $t_{1,7}$ | $t_{1,8}$ | $t_{1,9}$ | $t_{1,10}$ | $t_{1,11}$ | $t_{1,12}$ |
| $t_{1,13}$ | $t_{1,14}$ | $t_{1,15}$ | $t_{1,16}$ | $t_{1,17}$ | $t_{1,18}$ |
| $t_{1,19}$ | $t_{1,20}$ | $t_{1,21}$ | $t_{1,22}$ | $t_{1,23}$ | $t_{1,24}$ |
| $t_{1,25}$ | $t_{1,26}$ | $t_{1,27}$ | $t_{1,28}$ | $t_{1,29}$ | $t_{1,30}$ |

| | | | | | |
|---|---|---|---|---|---|
| $t_{2,1}$ | $t_{2,2}$ | $t_{2,3}$ | $t_{2,4}$ | $t_{2,5}$ | $t_{2,6}$ |
| $t_{2,7}$ | $t_{2,8}$ | $t_{2,9}$ | $t_{2,10}$ | $t_{2,11}$ | $t_{2,12}$ |
| $t_{2,13}$ | $t_{2,14}$ | $t_{2,15}$ | $t_{2,16}$ | $t_{2,17}$ | $t_{2,18}$ |
| $t_{2,19}$ | $t_{2,20}$ | $t_{2,21}$ | $t_{2,22}$ | $t_{2,23}$ | $t_{2,24}$ |
| $t_{2,25}$ | $t_{2,26}$ | $t_{2,27}$ | $t_{2,28}$ | $t_{2,29}$ | $t_{2,30}$ |

Schedule for the first slice partition can be generated using ILP formulation III. But to get the schedule for the next slice partition using ILP formulation III,

we need to add the information about the schedule we get for the first one. This can be done by adding some additional constraints, these are:

$$st_{2,1} \geq st_{1,6} \tag{6.4.6.0.39}$$

$$st_{2,7} \geq st_{1,12} \tag{6.4.6.0.40}$$

$$st_{2,13} \geq st_{1,18} \tag{6.4.6.0.41}$$

$$st_{2,19} \geq st_{1,24} \tag{6.4.6.0.42}$$

$$st_{2,25} \geq st_{1,30} \tag{6.4.6.0.43}$$

$$\forall p_k \in P : \quad x_{(2,1)k} = x_{(1,6)k} \tag{6.4.6.0.44}$$

$$\forall p_k \in P : \quad x_{(2,7)k} = x_{(1,12)k} \tag{6.4.6.0.45}$$

$$\forall p_k \in P : \quad x_{(2,13)k} = x_{(1,18)k} \tag{6.4.6.0.46}$$

$$\forall p_k \in P : \quad x_{(2,19)k} = x_{(1,24)k} \tag{6.4.6.0.47}$$

$$\forall p_k \in P : \quad x_{(2,25)k} = x_{(1,30)k} \tag{6.4.6.0.48}$$

## 6.5 Experimental Results:

The results we have get applying ILP formulation III on video streams are presented in this section. We have applied ILP formulation III and ILP formulation III with overlapped slice partitioning, on same data slices; and the results for these two methods are shown below. Using the utilization and speedup achieved by the methods, we can figure out how effective the generated schedules are. The time to solve the constraints generating the schedule is also given here for both the methods.

The results given in the following figures are from applying the scheduling strategies on video stream of resolution 176x144 and frame rate 25fps containing 50 slices of dimension 5x11 (in no. of macroblocks). Number of cores available for scheduling is taken as four.

Figure 6.1 showing the speedup that can be achieved using the given schedule constructed by both the strategies for each slice. Figure 6.2 showing the average of the core utilizations for scheduling macroblocks of each slice. Figure 6.3 showing time to solve the ILPs for different slices using standard tools on a quad core machine with 8GB of RAM and 15GB swap space.

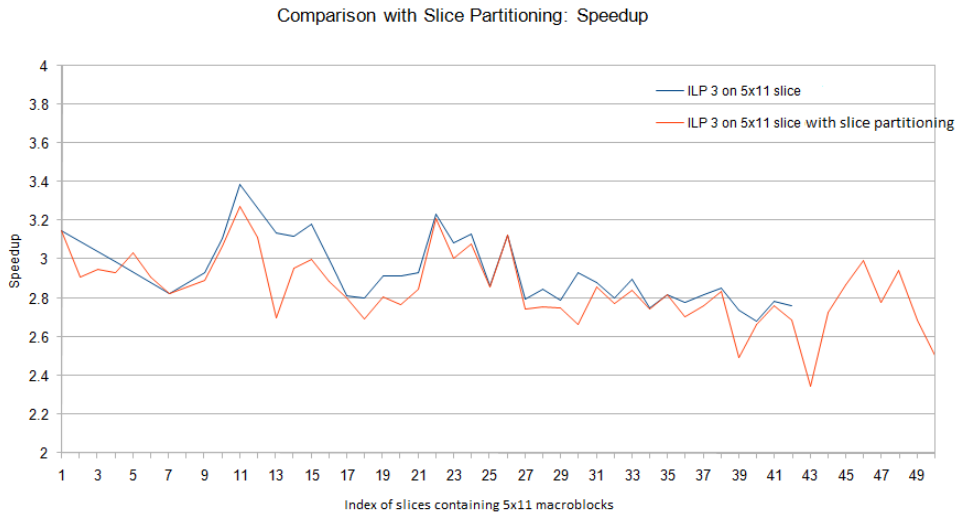



Figure 6.1: *Graph comparing speedup achieved by simple ILP formulation III and ILP formulation III with overlapped slice partitioning using 4 core*

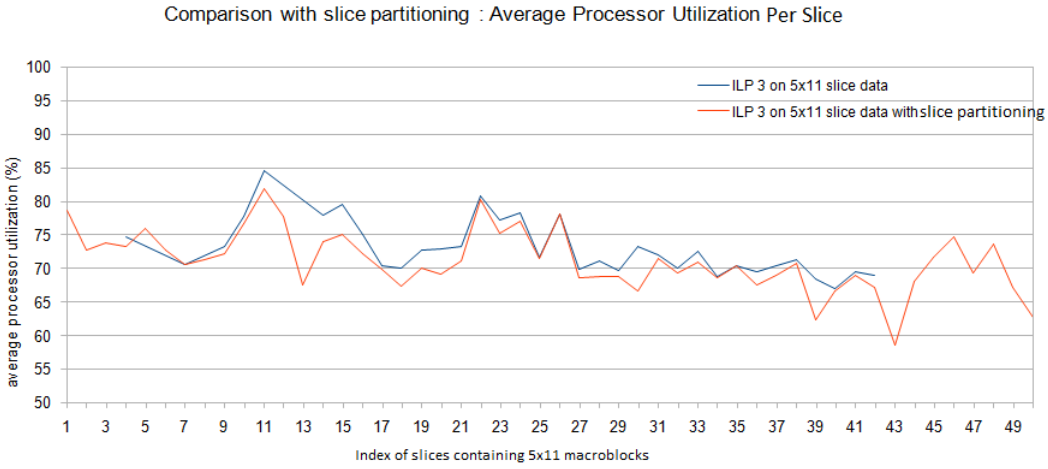



Figure 6.2: *Graph comparing average processor utilization achieved by simple ILP formulation III and ILP formulation III with overlapped slice partitioning*

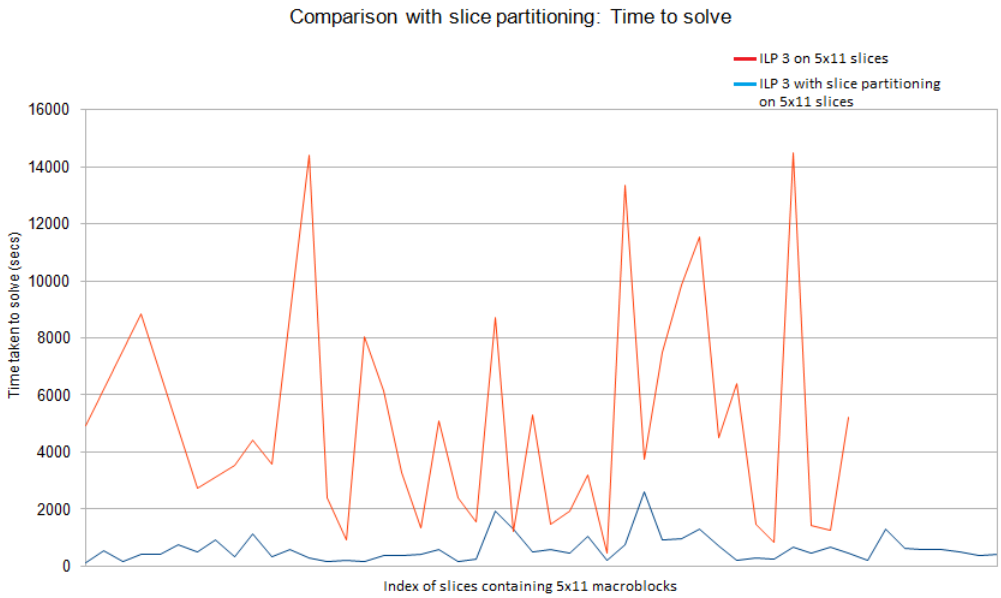



Figure 6.3: *Graph comparing time to get the schedule by simple ILP formulation III and ILP formulation III with overlapped slice partitioning*

# Chapter 7

# Conclusion and Bibliography

## Conclusion

The amount of computation in video processing will increase further in future standards. The number of processing cores in a computer is also increasing with days. To bridge them we need good schedulers to make these processing faster. Our work provides an ILP based optimized scheduling strategy for decoding H.264 videos in multicore processors. We have shown in an average 2.9 speedup can be achieved for a video stream. The ILP based scheduler can also be used to measure the performance of other greedy strategies which is much faster than this. For slices with higher dimensions we have given a strategy to partition it and use the ILP based scheduling strategies on each such partitions. So this scheduling strategy can be applied for decoding videos of various resolutions on a number of processing cores with good speedup and utilization.

# Bibliography


[1] Ian E. G. Richardson,*H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*,WILEY

[2] Jae-Beom Lee, Hari Kalva,*The VC-1 and H.264 Video Compression Standards for Broadband Video Services*,Springer

[3] ITU-T Rec. H.264, Version Mar. 2010, *Advanced Video Coding for Generic Audiovisual Services*

[4] Ying Yi, Wei Han, Xin Zhao, Ahmet T. Erdogan and Tughrul Arslan "An ILP Formulation for Task Mapping and Scheduling on Multi-core Architectures" Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.

[5] Meenderinck, C., Azevedo, A., Alvarez, M., Juurlink, B., Ramirez, A.:" Parallel Scalability of H.264." In: Proc. First Workshop on Programmability Issues for Multi-Core Computers. (January 2008)

[6] A. Azevedo, B.H.H. Juurlink, C.H. Meenderinck, A. Terechko, J. Hoogerbrugge, M. Alvarez, A. Ramirez, M. Valero, "A Highly Scalable Parallel Implementation of H.264," Transactions on HighPerformance Embedded Architectures and Compilers (HiPEAC), September 2009.

[7] A. Azevedo, C. Meenderinck, B. Juurlink, A. Terechko, J.Hoogerbrugge, M. Alvarez, A. Ramirez, and M. Valero, "Parallel H. 264 Decoding on an Embedded Multicore Processor," *Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers*, Springer, 2008, pp. 404-418.

[8] Jike Chong, N. Satish, B. Catanzaro, K. Ravindran, and K. Keutzer,"Efficient Parallelization of H.264 Decoding with Macro Block Level Scheduling," *Multimedia and Expo, 2007 IEEE International Conference on,* 2007, pp. 1874-1877.

[9] Shuwei Sun, Dong Wang, and Shuming Chen, "A highly efĭnĄcient parallel algorithm for H.264 encoder based on macro-block region partition," *Lecture Notes In Computer Science,* pp. 577âĂŞ585, 2007.

[10] Sihn, K., Baik, H., Kim, J., Bae, S., Song, H.: Novel approaches to parallel H. 264 decoder on symmetric multicore systems. In: Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing-Volume 00, IEEE Computer Society (2009) 2017âĂŞ2020

[11] Ahmet GÃijrhanl, Charlie Chung-Ping Chen, Shih-Hao Hung, "GOP-Level Parallelization of the H.264 Decoder without a Start-Code Scanner," *2nd International Conference on Signal Processing Systems (ICSPS),* 2010

[12] Mauricio Alvarez Mesa, Alex RamiÌĄrez, Arnaldo Azevedo, Cor Meenderinck, Ben Juurlink, Mateo Valero, "Scalability of Macroblock-level Parallelism for H.264 Decoding," icpads, pp.236243, 2009 15th International Conference on Parallel and Distributed Systems, 2009

[13] http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/